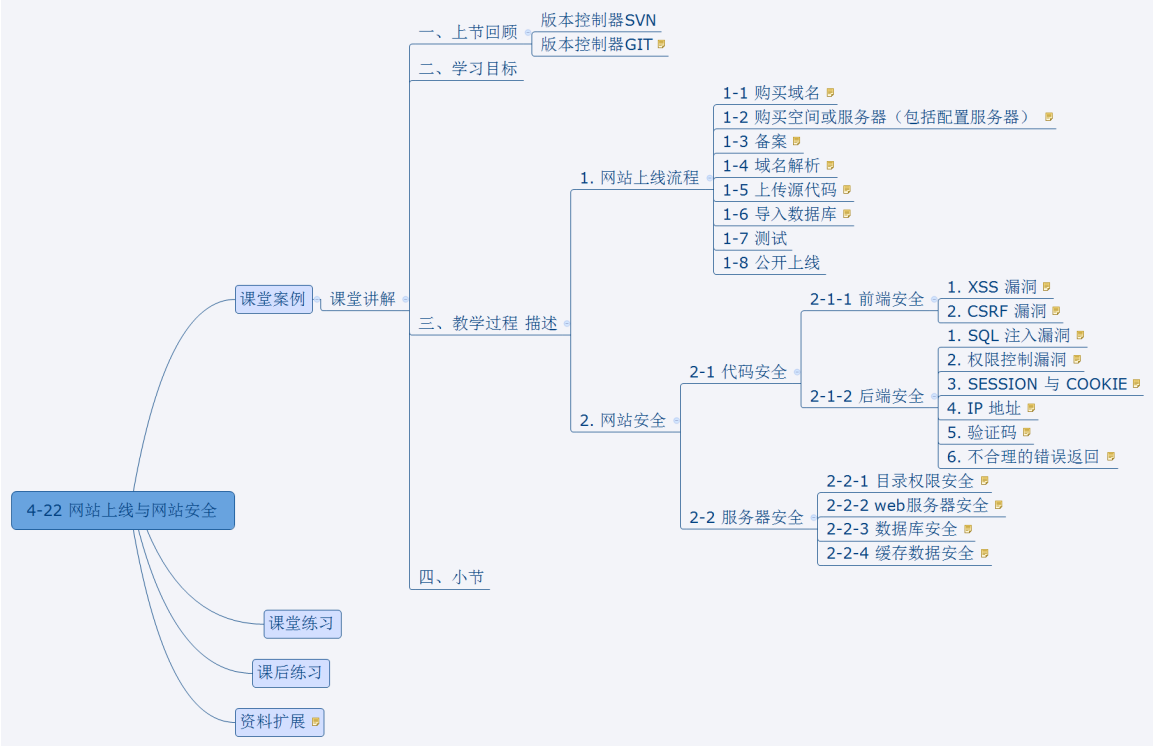
4-22 网站上线与网站安全



1. 课堂案例

课堂讲解

一、上节回顾

版本控制器SVN

版本控制器GIT

git init

git clone

git add

git status

git diff

git commit

git reset HEAD

git rm

git mv

二、学习目标

三、教学过程 描述

1. 网站上线流程

1-1 购买域名

购买域名其它是在网站开发的第一个流程，在我们开发前就应该要操作的步骤

1-2 购买空间或服务器（包括配置服务器）

目前主流的空间，分为虚拟空间，和云服务器，
一般企业网站，可以购买虚拟空间，如果是一些大型系统，或是平台，都是购买去主机，

区别：
虚拟空间为，固定配置，没办法定定义，有很多的限制，不能安装软件，空间较小，有一定的并发限制
，

云主机，可以按自己的需求，对服务器的软件进行安装，配置，可以完全自定义

1-3 备案

对于国内的主机，还需要对域名进行备案，比如将一个域名为 abcd.com
的域名绑定到一个国内的主机：如啊里云，这样就需要对域名进行备案，请意义，要备案是域名，不是主机。

备案流程：

不同的平台，稍有不同，但大概步骤都是一样的..如阿里去的备案流程如下：

备案流程看如下地址：https://help.aliyun.com/knowledge_detail/36922.html

备注：

为了顺利备案，需要准备大量的资料，所以资料越齐，速度越快，需要注意的是，首先，上传的文件要清晰，其次，如果是个人备案，只能备案博客类型的网站。不然审核很难通过。。

1-4 域名解析

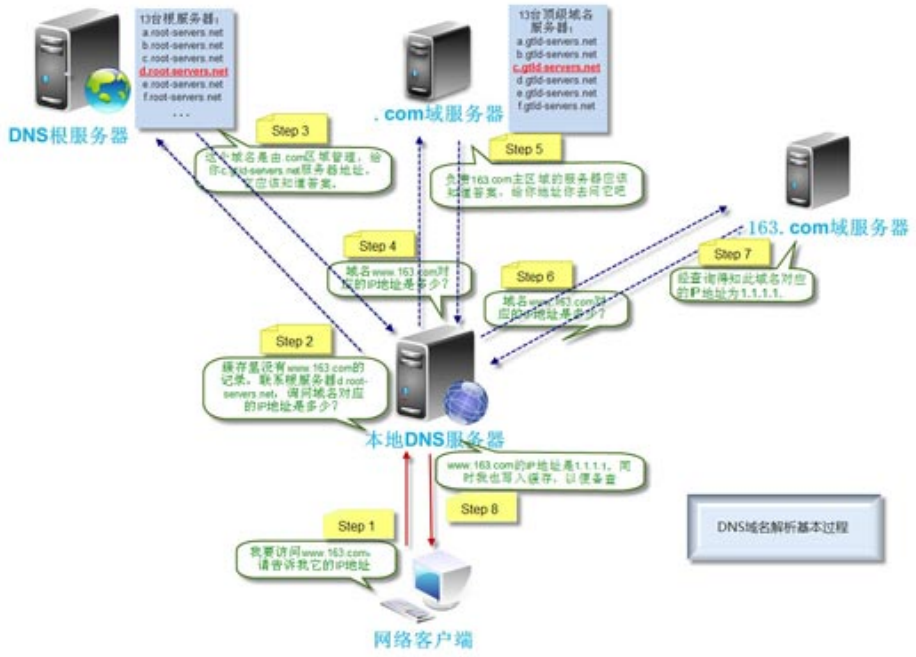
域名解析，目的是为了，使你购买的域名，对够对应到你的网站源代码是存放在哪个服务器IP的

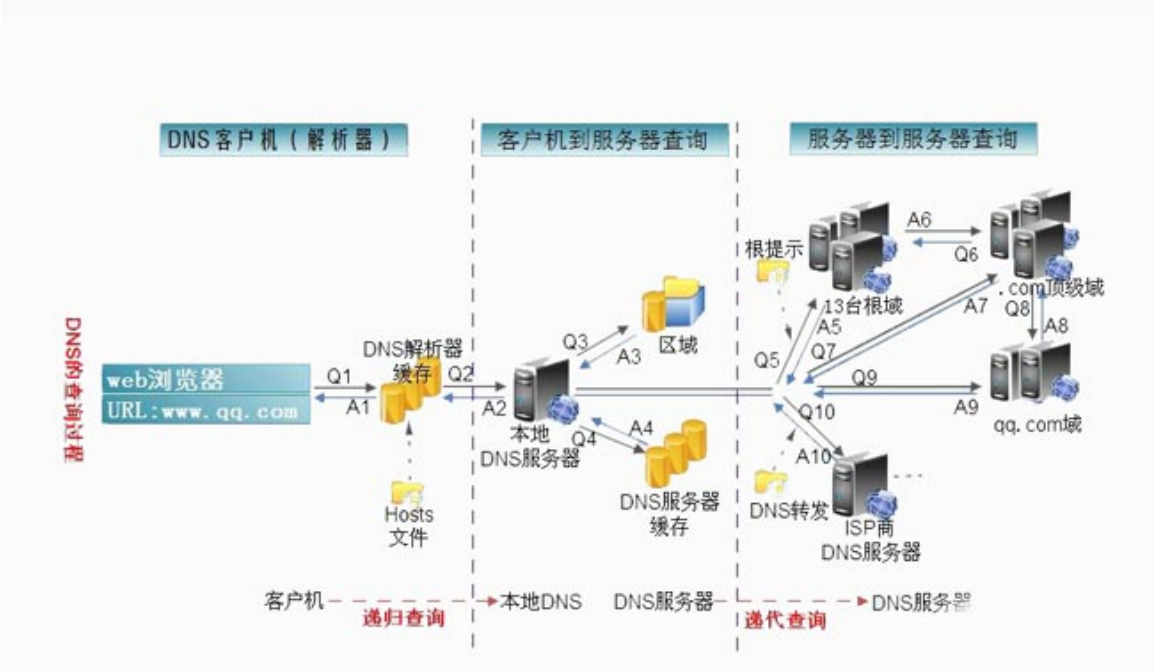
在此之前，我们先来了解下DNS..　　因为我们的域名解析就是将我们的解析规则更新到DNS服务器，同时我们在做多域名部署时，也需要理解DNS服务器..

DNS原理：

概念：DNS 服务器，域名，IP

查询原理图：





1-5 上传源代码

项目开发完成后，本地测试通过后，即可通过FTP软件，将本地代码上传到服务器空间中

1-6 导入数据库

使用工具，如: navcat 或作用命令行，将本地测试数据库数据，导入到线上数据库

1-7 测试

1-8 公开上线

2. 网站安全

2-1 代码安全

2-1-1 前端安全

1. XSS 漏洞

XSS 跨站脚本攻击，其实就是你的网站存在执行非法的JS脚本可能，即漏洞

我们来看个DEMO：

我们新建一个 xss.php文件 代码如下：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>XSS原理重现</title>
</head>
<body>
<form action="" method="get">
<input type="text" name="xss_input">
<input type="submit">
</form>
<hr>
<?php
if(isset($_GET['xss_input'])){
    $xss = $_GET['xss_input'];
    echo '你输入的字符为<br>'.$xss;
}
?>
</body>
</html>
```

功能很简单，就是打印出我们输入的内容：

这种方式，现在很多现代的浏览器已经可以过滤这种XSS攻击了...，

但对于一些旧的IE浏览器，是没办法过滤的...

常见的XSS案例，当然如下这种代码已经被现代的浏览器过滤处理无法执行了，但还是直得我们警醒。

1，直接使用JS脚本。

2，对JS脚本进行转码。

3，利用标签的触发条件插入代码并进行转码。

4，使用16进制来写(可以在傲游中运行)

<img STYLE=”background-image: /75/72/6c/28/6a/61/76/61/73/63/72/69/70/74/3a/61/6c/65/72/74/28/27/58/53/53/27/29/29″>

以上写法等于<img STYLE=”background-image: url(javascript:alert(‘XSS’))”>

XSS攻击解决办法

具体执行的方式有以下几点：

第一、在输入方面对所有用户提交内容进行可靠的输入验证，提交内容包括URL、查询关键字、http头、post数据等

第二、在输出方面，在用户输内容中使用<XMP>标签。标签内的内容不会解释，直接显示。

第三、严格执行字符输入字数控制。

四、在脚本执行区中，杜绝无用户输入。

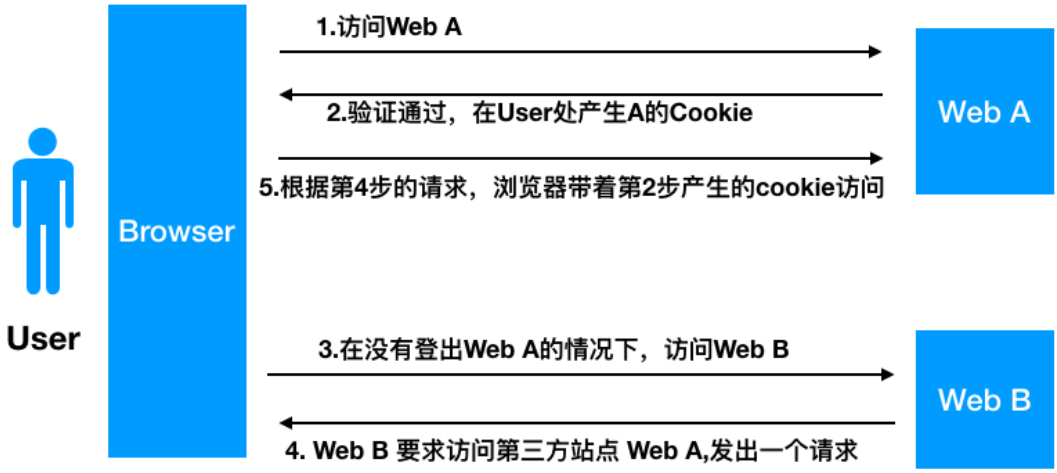
2. CSRF 漏洞

CSRF（Cross-site request forgery）跨站请求伪造，也被称为“One Click Attack”或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），

但它与XSS非常不同，XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

浏览器有一个特性：就是会自动带上当前网站的会话cookie，因此就会CSRF提供了机会：

原理如下：



6.最终，Web A不知道请求是User发出的，还是Web B发出的，由于浏览器会自动带上User 的cookie ，所以 Web A会根据用户授权处理第4步的请求，这样Web B就达到了模拟用户操作目的，想像下，如果Web A 是一个银行网站，这样 Web B 模拟User 转账出去，这是多么恐怖的事情

从上图可以看出，要完成一次CSRF攻击，受害者必须依次完成两个步骤：

□□1.登录受信任网站A，并在本地生成Cookie。

□□2.在不登出A的情况下，访问危险网站B。

□□

看到这里，你也许会说："如果我不满足以上两个条件中的一个，我就不会受到CSRF的攻击"。是的，确实如此，但你不能保证以下情况不会发生：

□□1.你不能保证你登录了一个网站后，不再打开一个tab页面并访问另外的网站。

□□

2.你不能保证你关闭浏览器了后，你本地的Cookie立刻过期，你上次的会话已经结束。（事实上，关闭浏览器不能结束一个会话，但大多数人都会错误的认为关闭浏览器就等于退出登录/结束会话了.....）

□□3.上图中所谓的攻击网站，可能是一个存在其他漏洞的可信任的经常被人访问的网站。

□□

上面大概地讲了一下CSRF攻击的思想，下面我将用几个例子详细说说具体的CSRF攻击，这里我以一个银行转账的操作作为例子（仅仅是例子，真实的银行网站没这么傻:>）

示例1：

□□

银行网站A，它以GET请求来完成银行转账的操作，如：http://www.mybank.com/Transfer.php?toBankId=11&money=1000

□□危险网站B，它里面有一段HTML的代码如下：

□□<img src=http://www.mybank.com/Transfer.php?toBankId=11&money=1000

□□

首先，你登录了银行网站A，然后访问危险网站B，噢，这时你会发现你的银行账户少了1000块.....

□□

为什么会这样呢？原因是银行网站A违反了HTTP规范，使用GET请求更新资源。在访问危险网站B的之前，你已经登录了银行网站A，而B中的<img>以GET的方式请求第三方资源（这里的第三方就是指银行网站了，原本这是一个合法的请求，但这里被不法分子利用了），所以你的浏览器会带上你的银行网站A的Cookie发出Get请求，去获取资源“http://www.mybank.com/Transfer.php?toBankId=11&money=1000”，结果银行网站服务器收到请求后，认为这是一个更新资源操作（转账操作），所以就立刻进行转账操作.....

示例2：

□□为了杜绝上面的问题，银行决定改用POST请求完成转账操作。

□□银行网站A的WEB表单如下：□□

```
□□<form action="Transfer.php" method="POST">
□□□□<p>ToBankId: <input type="text" name="toBankId" /></p>
□□□□<p>Money: <input type="text" name="money" /></p>
□□□□<p><input type="submit" value="Transfer" /></p>
□□</form>
```

□□后台处理页面Transfer.php如下：

```
□□<?php
□□□□session_start();
□□□□if (isset($_REQUEST['toBankId'] &&□isset($_REQUEST['money']))
□□□□{
□□□□    buy_stocks($_REQUEST['toBankId'],□$_REQUEST['money']);
□□□□}
□□?>
```

□□危险网站B，仍然只是包含那句HTML代码：

```
□□<img src=http://www.mybank.com/Transfer.php?toBankId=11&money=1000
□□
```

和示例1中的操作一样，你首先登录了银行网站A，然后访问危险网站B，结果.....和示例1一样，你再次没了1000块～T_T，这次事故的原因是：银行后台使用了$_REQUEST去获取请求的数据，而$_REQUEST既可以获取GET请求的数据，也可以获取POST请求的数据，这就造成了在后台处理程序无法区分这到底是GET请求的数据还是POST请求的数据。在PHP中，可以使用$_GET和$_POST分别获取GET请求和POST请求的数据。在JAVA中，用于获取请求数据request一样存在不能区分GET请求数据和POST数据的问题。

示例3：

□□

经过前面2个惨痛的教训，银行决定把获取请求数据的方法也改了，改用$_POST，只获取POST请求的数据，后台处理页面Transfer.php代码如下：

```
□□<?php
□□□□session_start();
□□□□if (isset($_POST['toBankId'] &&□isset($_POST['money']))
□□□□{
□□□□    buy_stocks($_POST['toBankId'],□$_POST['money']);
□□□□}
□□?>
```

然而，危险网站B与时俱进，它改了一下代码：

```
<html>
□□<head>
□□□□<script type="text/javascript">
□□□□□□function steal()
□□□□□□{
        □□□□ iframe = document.frames["steal"];
□□    □□     iframe.document.Submit("transfer");
□□□□□□}
□□□□</script>
□□</head>

□□<body onload="steal()">
□□□□<iframe name="steal" display="none">
□□□□□□<form method="POST" name="transfer"□action="http://www.myBank.com/Transfer.php"
□□□□□□□□<input type="hidden" name="toBankId" value="11">
□□□□□□□□<input type="hidden" name="money" value="1000">
□□□□□□</form>
□□□□</iframe>
□□</body>
</html>
```

如果用户仍是继续上面的操作，很不幸，结果将会是再次不见1000块.....因为这里危险网站B暗地里发送了POST请求到银行!

□□

总结一下上面3个例子，CSRF主要的攻击模式基本上是以上的3种，其中以第1,2种最为严重，因为触发条件很简单，一个<img>就可以了，而第3种比较麻烦，需要使用JavaScript，所以使用的机会会比前面的少很多，但无论是哪种情况，只要触发了CSRF攻击，后果都有可能很严重。

□□

理解上面的3种攻击模式，其实可以看出，CSRF攻击是源于WEB的隐式身份验证机制！WEB的身份验证机制虽然可以保证一个请求是来自于某个用户的浏览器，但却无法保证该请求是用户批准发送的！

CSRF的防御

CSRF的防御可以从服务端和客户端两方面着手，防御效果是从服务端着手效果比较好，现在一般的CSRF防御也都在服务端进行。

服务端进行CSRF防御

服务端的CSRF方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。比如，图片验证码，表单令牌难，及表单隐藏随机数

2-1-2 后端安全

1. SQL 注入漏洞

SQL注入的概述

SQL注入即是指web应用程序对用户输入数据的合法性没有判断，攻击者可以在web应用程序中事先定义好的查询语句的结尾上添加额外的SQL语句，以此来实现欺骗数据库服务器执行非授权的任意查询，从而进一步得到相应的数据信息。

SQL注入原理：

假设：我们有一个登录的表单页面，如下：

从前面我们学的知识，我们可以很快就能写出实现登录的SQL语句：如下

```php
$uname = $_POST['uname'];
$pass = $_POST['upass'];

$sql = "SELECT * FROM `user` where uname='".$uname."' and upass='".md5($pass)."'";
```

执行上面的sql 语句，如果查询到数据，表示登录成功。

假设我们 在 User ID 中输入：admin， 在Password 中输入 : admin

这样我们先得到 这样一条sql语句:

```
$sql = "SELECT * FROM `user` where uname='admin' and upass='21232f297a57a5a743894a0e4a801fc3'";
```

再来设想下：我们如何这样填写表单：

这样一来我们得到的sql语句将会是如下的形式：

```
$sql = "SELECT * FROM `user` where uname='' or 1=1-- ' and upass='21232f297a57a5a743894a0e4a801fc3'";
```

红框中是我们输入的内容，其中的 -- 是注释，意味着--后面的都是没效内容，即不检查密码了，同时同上于前面的sql 语句有一个 or 1=1 这样的语句无论如何都是成立的，此时SQL语句变成了：

$sql = "SELECT * FROM `user` where uname='' or 1=1

这条SQL将会把整个user表的数都给查询出来，因此，我们的系统会判断登录成功！

这样我们就实现了匿名登录。

是不是很可怕，我们随随便便就可以登录成功，而且这还是最简单的注入，SQL语句千变万化，只要整个网站系统，有一个地方有这样的漏洞，就可能被利用，导致整个数据库的数据被偷盗。

SQL注入漏洞，不是只有表单才存在，URL地址也可以成为SQL注入的入口，只要这漏洞存在，那么可以构造很多的SQL语句。

我们的课程不是网站的攻击课程，因此不会去细致的讲解如何攻击一个网站，或是如何去盗取一个网站的数据库资料，因此点到为至，我们要讲的话，如何防止这样的问题出现，做到我们开发的网站不会存在这样的漏洞。

其次，我们在一此数据库操作时，页面会弹出如下形式的错误：
Column not found: 1054 Unknown column 'lname' in 'field list'

这种错误提示也有可能会暴露了我们的数据库的信息。因此我们线上环境一般都是关闭错误详情的。只能看到一个500错误，或是一个公用的错误提示页面。

如何预防SQL注入：
一、 服务器的安全设置(php.ini 配置文件)
(1) 打开php的安全模式 safe_mode = on
(2)关闭危险函数 如phpinfo()
disable_functions = system,passthru,exec,shell_exec,popen,phpinfo
(3)关闭PHP版本信息在http头中的泄漏
expose_php = Off
(4)关闭注册全局变量
register_globals = Off
(5)打开magic_quotes_gpc来防止SQL注入
magic_quotes_gpc = On
(6) 关闭错误信息
display_errors = Off
(7)
由于关闭了错误信息，因此我们需要打开错误日志，以便发生错误时，我们去查询错误的原因
log_errors = On
error_log = D:/usr/local/apache2/logs/php_error.log

二、程序编写安全

永远不要相信用户，所有用户输入的数据都进行过滤，比如，对长度检查，数据类型检查，过

滤特殊字符如，前面我们讲到的 单引号 及 or 1=1 这种类似的字符，当然我们现在的mysqli ,pdo 其实系统都会帮我们过滤这种，但做为一个专业的程序人员我们需要有这样的习惯跟意识

过滤用户输入的像：select update drop insert 等关键字

另外，我们目前的一些流行框架，都有帮我们做这相防止注入的工作，我们可以选择一个流行的框架来操作我们的数据库，这样可以提高安全性。。

2. 权限控制漏洞

对于一些有很多权限的的WEB系统，我们推荐使用最小权限原则，比如，管理文章的，决不让用户有其它权限，比如用户管理，免得一些恶意的用户，权限过大，导致系统的一些关键信息被盗取而被恶意使用

3. SESSION 与 COOKIE

session 与 cookie 经常被用来存取会话数据，其也存在安全漏洞，比如我们前面讲的CSRF就是利用了浏览器cookie原则

恶意用户会通过伪造表单，及cookie
的方式对web服务器进行访问，同时对于一些未加密的cookie信息，也很容易被泄漏，比如在网吧登录了某个网站，因为此网站未对cookie信息进行加密，造成cookie信息以明文的方式，存储在网吧电脑的硬盘上，当下一位用户，如果使用一些工具，很轻意就能将这些信息提取出来...
然后用你的数据进行登录，如果是游戏账号，或是银行账号，那该是多么恐怖！！

如何防止：

1. 对session 或 cookie 加密存储，
2. 对于重要的信息，不要存储在cookie上，可以选择session存储，因为session相对cookie还是要安全一些
3. 设置合理的有效期

4. IP 地址

IP地址防御，主要是针对采集做的安全限制，比如服务器检查到，一段时间内，一个IP持续性的访问我们的网站，而且很频繁，这个时候就要去分析是不是采集程序，在对我们的站点进行采集操作，或者大连的请求都是访问我们网站的图片信息，这个时候可以考虑，我们站点的图片是不是被盗用了

防御：

1.
时刻关注访问日志，如何发现网站突然变慢，马上检查访问日志，看是否存在异常，当然也可以开启IP检测，（但这会增加服务器的负担）系统检测到一段时间内超过我们设置的访问量，就进行限制，比如当设置，1秒钟访问了站点100个页面，我们就断定为采集，这时候我们可以采取，限制访问，或是通过一定的难手段才让其访问，如需要输入图弄难码才让其访问！

2.
开启图片防盗连，这一般是针对大型系统，系统里大量的图片，这时候总会有一些其它的站点来盗用我们的图片连接，这时我们需要借专门的工具来进行防盗连，如我们可以使用啊里的OSS存储系统来，存储我们的图片，因为OSS提供了防盗连功能，我们自己无需再去那一套系统来防盗连

5. 验证码

验证码是我们在WEB开发中用得最多的的安全手段：

比如我们登录需验证码，发送手机短信也需要验证码

验证码又分为：
1. 图片难码
2. 手机难码
3. 邮箱验证码

使用验证码可以防止，有人使用程序来操作我们的请求，这样可以有效防止恶意的请求，比如京东的注册页面：

我们看到，我们需要输入手机号，然后是图片难码，最后才是手机验证码，而且，它会首先验证手机号，及图片难码是否正确，才会触发获取手机难证码的请求。
我们思考下它为什么要这么做？？

手机号肯定是要的，不然我系统怎么知道短信要发到那个手机号码。

可是为什么要先正确输了图片验证码才能发送手机验证码呢？？？

原因如下：

假设我们可以直接输了手机号码就直接点击：获取难证码 按钮，这样就发送了一条短信。。。

那么，如果我就京东的竞争对手，也是一个坏人，我发现京东，可以使用这个接口发送手机短信：我可以这么做，我写一个程序：

定义一个 大型数组：比如有10万个元素的数组，每个元素就是一个手机号码：如下

```
$arr  = array(
   '134587876565',
   '135987876549',
   .....这里有一万个手机号码

);
```

然手循环这个数组调用京东的短信接口：发送10万条短信到这些手机上。如果京东的后台没有做其它的限制，那么我这10万条短信就发出去了。

大家可能会问我发这10万条短信有意义吗？
我想说的是，对我来说没有任务意义，但对于京东可是损失了真金白银。。发送短信可都是要钱的，做为竞争对手的我来说，我可以消耗竞争对手的金钱，对于我来说就是有利的...

当然，京东不会么傻B，他首先让你输图片难码，输对了才能发送点击按钮，这样我们的程序没办法识别图片难码里的内容，也就无法通过程序去输入正确的难码了，短信肯定也就不能通过程序发送出去了，其次，京东也做了检查，即使你难码正确，你也不能无限制的发送，他会限制，一个手机号码只能发送几条....

但从这点，我们知道，我们明白，什么样的场合需要设置图片难码，以及需要限制发送短信的条数，

老师曾经就吃过这样的夸，一个注册页面，因为安全没做到位，一个晚上，被用户恶意的消费了5000多元的短信费用，还好发现得早，不然后果.....

6. 不合理的错误返回

错误返回，就是不让用户看不到不应该看到的内容，比如前面提到的网站的错误信息，或是sql错误信息

2-2 服务器安全

2-2-1 目录权限安全

目录权限，就是某些目录，不让用户有写权限，或是执行权限，比如一些config
目录，设置为只读，这样，即使恶意用户通过一些途径可以操作php代码来改写数据，但由于当前目录是只读，那么他也就束手无策，再比如说，用户获得了某些php的改写权限，他想上传一个恶意文件到某个目录，如果当前目录为只读，那么他的上传操作将不会成功

2-2-2 web服务器安全

服务器安全，主要指的是配置安全，如防火墙，IP黑名单，php.ini中的一些安全配置等

2-2-3 数据库安全

数据库安全，指的是数据库服务器的配置安全，如允许那些IP连接数据库，即白名单，是否允许远程登录，什么样的用户，对数据库有什么样的操作权限，比如当前连接的用户，对哪此数据库有操作权限，是什么样的操作权限，增删改查都有权限，还是只允许查权限等

2-2-4 缓存数据安全

缓存，主要是指的是文件缓存，或内存缓存安全问题，文件缓存如果是明文方式，比如，我把用户名密码，以明文的方式存在我们的服务器文件缓存中，如果服务器被攻破，那么攻击者，直接打开缓存文件就能看到里面的用户密码信息！

内存缓存：我们知道memcache是没有用户验证的机制，如果用户通过一些途径知道了memcache的的IP地址，及商品，同时当前memcache又没有做安全限制，比如没有做防火墙，同时此memcahce是在公网上，那可就糟糕了，所有数据暴露了..

防御：

文件缓存：进行加密

内存缓存：如memcache

部署到内网，或是通过防火墙加白名单，只请允许我们自己的WEB服务IP访问，其它IP禁止访问，同时对敏感数据加密

四、小节

2. 课堂练习

3. 课后练习

4. 资料扩展

相关文档：http://netsecurity.51cto.com/art/201108/287651.htm