软件系统安全脆弱性测试技术*)

Software System Security Vulnerability Testing Technologies

王 航 戴英侠 单国栋 连一峰

(中国科学院研究生院信息安全国家重点实验室 北京100039)

Abstract The current system security research focuses on defence technology, but it is not enough. Based on in-depth comprehension of system attack methods, we can build vulnerability model of software, then rely on statically source code scanning, environment fault injection and disassembly technologies, we can help find and locate unknown vulnerabilities, which give us advantage in future information antagonizing.

Keywords Vulnerability, Software testing, Vulnerability model

1. 前言

目前网络安全的研究重点在于防御技术,但是仅仅依靠被动的防御是不够的,实践表明,攻防的焦点往往集中在对漏洞的发现、利用和检测上,通过对漏洞和弱点的原理分析,建立一个能够指导实践的脆弱性模型,并在此基础上研究主动发现系统安全隐患的技术,可以改变被动防御的局面,在信息对抗中掌握主动。这项研究的难点在于,它必须建立在对网络攻击方法的深入了解的基础上,而对于系统攻击的理论和技术,对于安全漏洞产生的机制、触发漏洞所需要的条件等问题长期以来没有得到学术界足够的重视。

从国外的研究情况来看,美国一向重视计算机漏洞的研究工作。从70年代中期开始,南加州大学、美国国家标准局、美国海军研究实验室(Naval Research Laboratory)等机构启动了一系列关于操作系统和软件脆弱性分析的研究项目。其中普度大学 COAST 实验室在计算机漏洞的研究方面一直处于领先地位。该实验室的 Aslam 和 Krsul 研究了漏洞的分类问题,加深了人们对于软件漏洞和安全机制的理解。他们把软件漏洞分为两大类:第一是由于程序的逻辑或设计错误而导致的漏洞;第二是由不适当的安装和配置而导致的漏洞。从方法论的角度讲,Krsul 提出了两种分类方法,一是后验分类,即通过经验的观察之后而得到的分类,二是先验分类,即非经验型的抽象模型分类,这种分类方式适合于对漏洞机制的理解,可以避免同样的漏洞再次出现。这些理论已经成为软件系统漏洞研究的理论基础。

2. 软件系统脆弱性模型

在对安全漏洞进行整理的过程中发现,漏洞的具体表现形式很多,如果不能构造一个抽象的脆弱性模型,就难以对漏洞产生的原因有深入的理解,发现和定位新的漏洞就会很困难。根据不同的抽象层次可以建立不同的模型,从不同的抽象层次来看,对同一个漏洞会有不同的理解,例如最常见的"缓冲区溢出"漏洞,从低层次上来说可能是参数验证错误;从高一些的层次看,这可以是一个同步或竞争条件错误;从更高的层次看,这是一个逻辑错误。

1993年,美国海军研究实验室的 Landwher 建立了如表1所示的分类模型,这个模型主要是根据对已知漏洞的归纳得到的,虽然在概念上有交叉和模糊的地方,但是它对于理解漏洞产生的根源是有帮助的。这个模型的主要缺点是可操作性不强,难以根据这个模型编写自动测试程序。

表1 Landwher 的漏洞分类模型

故意	特洛伊木马	不能复制
		能够复制(病毒)
	陷门	
	逻辑/时间炸弹	
	隐秘通道	存储
		定时
	其他	
无意	校验错误	
	范围错误	
	串行错误	
	认证/鉴别错误	
	边界条件错误(含资源耗尽合约束错误)	
	其他可利用的逻辑错误	

在研究过程中我们发现,相对于测试软件系统安全性这个目的来说,我们可以仅仅考虑由于逻辑错误导致的安全漏洞。为了具体操作的方便,又将逻辑错误分为三大类:①由于程序运行环境导致的漏洞;②由于常见编程疏忽导致的漏洞;③由于配置不当导致的错误。

显然,这种分类方法是不完备的,但它的优点在于可操作性比较强,对于这三类中的每一类别,我们可以方便地列出具体的漏洞表现特征。举例来说,对于由于程序运行环境导致的漏洞,可以对各种可能的环境特征进行假设,并通过程序环境特征的改变情况,分析漏洞形成的原因。每一个特征都可以使用"是"、"否"、"不适用"或"不知道"来表示,这些特征包括:

1)目录是否拥有特殊的权限?

2)在程序运行时用户是否能够看到程序的内部信息?

3)环境名称(如路径名)是否是特殊的系统对象?

4)在程序运行期间环境对象是否恒定不变(例如在程序运行时,其它任何主体是否能够改变这个对象)?

5)在程序运行时,对象是否存在?

6)在程序运行时,它所创建的临时文件是否能够被其它主体删除和窜改?

7)程序的运行是否总有足够的内存空间?

*)国家重点基础研究发展规划项目,项目编号:G1999035801,国家自然科学基金,项目编号:90104030。王 航 硕士研究生,主要研究方向为安全操作系统、入侵检测和扫描技术等。戴英侠 教授。单国栋 硕士研究生。

8)通过网络收到的数据是否总是有效、有界?

9)在环境变量中的数据是否总是有效、有界?

10)用户提供的输入数据是否总是有效、有界?

11)从其它文件输入的数据是否总是有效、有界?

12)由碎片重新组装成的数据对象是否改变了原对象的本质特性?

13)程序是否假设在特殊的路径中运行?

14)程序员是否假设对象的某个属性有预先规定的值。

15)程序数据的修改(通过外部主体)是否会影响程序的语义?

16)在创建一个文件时,任何已存在的同名文件是否能够被覆盖重写?

17)对对象的操作是否有足够的权限?

18)对象的名称和性质是否相关?

19)有特定名字的对象是否会仅仅由于其名字的原因而不被系统中的其它实体使用?

20)网络对象是否可信?

对于这二十种特征表现形式,我们都可以找到具体的漏洞与之相对应,这样的对应关系在寻找新的安全漏洞时能够提供很多方便。

3. 软件系统脆弱性测试

应该说,漏洞的发现和利用是一种充满了技巧和需要丰富经验的技术,但是这种技术并不是不可琢磨的,在发现漏洞的过程中仍然充满了很强的规律性,我们将测试方法分为源代码扫描、环境错误注入、反汇编三类,如图1所示。

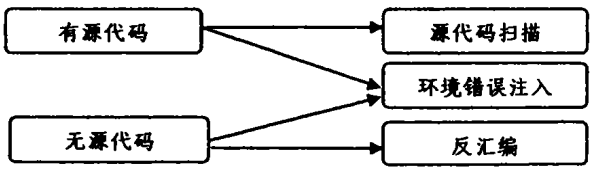

图1

3.1 静态源代码扫描

这方面比较早的工作是由美国加州大学 Davis 分校的 M. Bishop 和 M. Dilger 发表的论文:"Checking for Race Conditions in Unix File Access",他们实现了一个原型系统能够检测到一些常见的 C 和 C++编程漏洞,但是这个系统基本上是建立在 grep 这样的正则表达式匹配程序上的,所以会产生大量的虚报;此后,一个著名的安全组织 l0pht 开发了 slint,这个程序能够扫描大约100种常见的 C 和 C++编程漏洞;随后一个影响比较大的系统是 ITS4,这个程序与 slint 相似但是更加准确;普林斯顿大学的 John Viega,Tom Mutdosch 开发了扫描 java 源程序的系统,能够扫描12大类的 java 编程漏洞。这些都是比较通用的扫描程序,专门的扫描集中在对缓冲区溢出漏洞的发现上,这方面的一篇经典文献是 D. Wagner 等发表在 Proc. Network and Distributed Systems Security Symposium(NDSS 2000)上的"A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities"。

以 C 语言为例。一般都认为,C 语言是一种灵活而容易出错的语言,例如 glibc 的标准函数如 gets,sprintf,strcpy,strcat,system,popen,sprintf,甚至包括一般认为安全的函数 strncpy 和 strncat,如果使用不当都可能出现安全问题。问题在于:尽管这类安全漏洞一再出现,但是由于大多数程序员并不了解安全编程的知识,或者由于疏忽,对于这些函数的不安全使用在各种应用程序中仍然大量使用,即使是微软这样专业的大公司,其流行的 IIS 服务器仍然多次出现类似的问题。编程中常常出现的另外一个漏洞被称为"time-of-check-time-of-use",也就是说一个进程首先检查一个对象是否存在,或者检查对象的内容,接着读写这个对象,这里就包含一个假定,从检查到实际读写这个过程中对象不会被改写,但是这个假定不总是成立的,许多符号连接漏洞都出在这里。

正是由于相当多的安全漏洞在源代码级别上存在很强的类似性,因此源代码扫描的任务就是找出这样的代码段。最原始的方法就是在程序中查找那些容易出错的函数,但是很显然这会造成大量的虚报,国外的研究表明,如果不考虑函数调用的参数和调用环境,如果不对源代码进行词法分析和语法分析,就难以准确地把握程序的语义。源代码扫描的一个简单实现方法是参照安全编译器的做法,在词法分析和语法分析的过程中,对程序进行强约束的安全检查。这可以通过修改 gcc 源代码的方法来实现。

3.2 环境错误注入

考虑到程序执行是一个动态过程这个特点,显然静态的代码扫描是不完备的。由于某些环境因素导致的安全漏洞是大量存在的。例如:dtmail 是 Solaris CDE 所带的一个邮件客户端工具,它缺省设置了 setgid 属性,如果将 HOME 变量设置为一个超过372字节的字符串,那么堆栈中的缓冲区将会溢出,攻击者可以覆盖相邻缓冲区的某些数据,如果小心地构造一个数据结构,可能使程序调用一个攻击者指定的函数指针,从而攻击者可以以 mail 组权限执行任意代码。

环境错误注入是一种自动化的软件测试方法,这种方法在协议安全测试等领域中都已经得到了广泛的应用。在这方面的一篇经典文章是普度大学 WenLiang Du 和 Aditya P. Mathur 的"Vulnerability Testing of Software System Using Fault Injection",这种方法认为"系统"由"应用程序"和"运行环境"组成,所有的被认为不属于运行程序的代码就属于环境。例如:应用程序使用的环境中声明的全局变量或者文件和网络等公共资源。程序员总是假定他们的程序会在正常的环境中运行。当这些假设成立时,程序当然是正确运行的。但是,由于作为共享资源的环境,常常被其它主体所影响,尤其是恶意的用户,这样,程序员的假设就可能是不正确的。程序是否能够容忍环境中的错误是影响程序健壮性的一个关键问题。错误注入方法通过选择一个适当的错误模型试图触发程序中包含的安全漏洞。

错误注入需要选择一个适当的错误模型。这种模型的选择依赖于错误的本质。总的来讲,环境错误在两个方面影响应用程序。第一,应用程序从其环境中接收输入。环境错误变成输入错误,这个错误被应用程序的内部实体继承,有些应用程序不能正确地处理这种错误;第二,环境错误并不通过内部实体影响应用程序,而是和环境实体一起,当应用程序和环境发生相互作用时,如果不能正确处理好这些错误,就会触发违背安全策略的事件。在这种情况下,环境错误是导致安全漏洞的直接原因,环境错误的媒介是环境实体自己。对应于这两种方法,可以有直接和间接这两种环境错误注入方法。错误注入技术要解决的另一个关键问题是定位,即在测试系统中,在什么位置注入错误的问题。

环境错误注入的具体方法很多,举例来说,在读写文件时改变文件名的长度、使用相对路径和绝对路径、在名字中插入

"|","`","&"等对操作系统有特殊意义的字符,在读取网络数据时改变数据包的长度,使用坏格式的数据包,在网络协议中通过忽略协议步骤、增加额外步骤、重新排列等方法故意违反协议规则等。一种简单的环境错误注入方法是随机地产生一些长的字符串作为应用程序的输入,如果程序不能正确地处理长字符串输入,就可能导致系统崩溃。这种方法虽然简单,但是由于使用C语言很容易产生指针越界错误,因此这类漏洞即使在许多成熟的系统中也是存在的。例如一种比较流行的商用防火墙NetScreen的某些版本就有这种类型的漏洞,NetScreen提供了一个Web管理界面,管理员可以使用浏览器来进行防火墙配置。然而,由于缺乏对用户输入URL的长度检查,如果攻击者提供一个超长的URL,可能导致一个缓冲区溢出的发生,防火墙将崩溃,攻击方法非常简单,只要向80端口发送一个URL请求:"GET A…A(1220个A)HTTP/1.0\n\n",这时所有通过防火墙建立的连接都被中断,防火墙也不再响应任何新的连接请求。这种方法的优点是适用范围广,只要给定一个协议的格式,系统自动产生各种长度的输入串进行循环测试,使用人员不需要任何专业的知识。目前已经有一个称为ntomax的自由软件可以进行这种测试。它的缺陷也是很明显的,首先,必须预先知道协议的格式;其次,如果漏洞的触发需要同时满足多个条件,单一的输入长字符串不能找出漏洞;第三,这种测试方法不能发现协议中隐藏的后门;第四,如果系统具有崩溃检测的功能,在崩溃之后自动重启,难以确认测试的结果。

在具体实现方法上,由于系统与环境的交互一般都需要使用一些系统提供的系统调用,我们采用了挂接(hook)系统调用的方法,例如对文件系统来说,在Linux环境下使用可加载内核模块(Loadable Kernel Module)挂接open,execve,read,write等关键系统调用,在程序访问文件之前可以改变访问对象的某些属性,实验已经证明这种方法是有效的。

3.3 反汇编代码扫描

反汇编无疑是一种复杂的技术,但是对于不公开源代码的程序来说这往往是最有效的发现安全漏洞的办法,分析反汇编代码需要丰富的经验,因此不可能有一种完全自动的工具来完成这个过程。但是如果有一种方便的辅助工具来帮助简化这个过程,无疑是很有价值的,在今年的BlackHat大会上有一篇很好的讲演:"Auditing Closed-Source Applications—Using reverse engineering in a security context"。作者提出利用IDA(一种优秀的反汇编程序)提供的脚本语言来识别一些可疑的汇编代码序列。我们已经验证这种方法是有效的。

通过反汇编来寻找系统漏洞的好处是从理论上讲,不管多么复杂的问题总是可以通过反汇编来解决,它的缺点也是显然的,这种方法费时费力,对人的要求很高,由此而引申出一个问题,能不能采用半自动的方法,使用一种脚本语言来扫描汇编代码,帮助识别一些常见的编程漏洞,这就可以大大节省所需的时间,根据搜索深度的不同也能够识别一些相对较为复杂、触发条件较多的漏洞。经过实验证明这样的想法是可行的。

程序在运行过程中需要通过系统调用(API)来与操作系统和环境交互,这些API代码可以采用静态或动态链接的方法来调用,好的反汇编程序都能够很好地识别这些系统调用,在这方面做得最好的就是IDA(The Interactive Disassembler);此外,大多数程序都是使用少数几个通用编译器来编译的,在Windows下例如Visual C++,C++ Builder等,在UNIX下例如gcc,编译器在将C语言语句编译成汇编语言的时候是相当规律的。举例来说,strcpy(dst,src)总是编译成下面的汇编语句:

```
lea eax,[ebp+arg_1]
push eax
lea eax,[ebp+var_1]
push eax
call strcpy
```

其中arg_1表示一个传入的参量,var_1指向一个栈中的缓冲区,反汇编程序可以检查下面两个方面的问题,第一,dst是否指向堆栈中一段固定长度的缓冲区,src是否指向一个字符串变量而不是一段固定长度的字符串?通过这种检查可以比较精确地定位可能存在问题的strcpy调用。

再举一个例子来说,下面的这段汇编代码可能也是有问题的。

```
lea eax,[ebp+var_1]
push eax
mov eax,[ebp+var_2]
push eax
call ds:wsprintfA
```

可以看出,在调用wsprintf的时候格式化字符串是由一个变量提供的,如果这个变量指向的地址空间可以由用户控制,就存在格式化字符串攻击的可能性。

攻击者在寻找系统漏洞的时候,也会经常采用类似的办法来寻找可能的攻击点,既然存在这样的规律,那么完全有可能编写一些脚本扫描工具来帮助我们加快这个过程。在国外的一次实验中,通过扫描NetScape iWS 4.1中SHTML.DLL这个文件,在很短的时间内就定位了一个256字节的缓冲区溢出攻击,如果完全依赖人工检查的办法通常需要好几天的时间。

小结 软件系统的漏洞测试是在脆弱性模型指导的基础上,静态分析和动态测试相结合的过程。传统上这完全是凭借个人的经验手工完成的。我们的研究表明,采用一些半自动的方法,设计一些工具程序来指导和加快这个过程是有效的。软件漏洞的研究对于消除软件安全隐患,提高系统强健性都将起到重要的作用。

参考文献

1 BugTraq, r00t. Smashing The Stack For Fun And Profit. Phrack Magazine, 1996, 7(49)

2 Dark spyrit AKA Barnaby Jack. Win32 Buffer Overflows (Location, Exploitation and Prevention). Phrack Magazine, 1999, 9(55)

3 Scut. Exploiting Format String Vulnerabilities. TESO Security Group. http://www.team-teso.net, March 17, 2001

4 Flake H. Auditing binaries for security vulnerabilities. http://www.blackhat.com-/presetations/ bh-europe-00/HalvarFlake/HalvarFlake.ppt.

5 Du Wenliang, Mathur A P. Vulnerability Testing of Software System Using Fault Injection:[Coast TR 98-02]. 1998

6 Wegner D, et al. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. The 2000 Network and Distributed Systems Security Conf. San Diego, CA, Feb. 2000

7 Ghosh A K, O'Connor T, McGraw G. An automated approach for identifying potential vulnerabilities in software. In: Proc. IEEE Sympo. on Security and Privacy. May 1998

8 Evans D, et al. LCLint: a tool for using specifications to check code. SIGSOFT Symp. on Foundations of Software Engineering. Dec. 1994