Java入门---缓冲区溢出编程心得 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E5_85_A5_E9_97_A8_c97_138392.htm 前言:网上关于缓冲区溢出的资料也有很多,但我在阅读过程中发现介绍的都不是很明了,而且各网站也只是转贴老外的那篇译文而已,不仅内容有缺损,而且程序也无法调通,因为GCC版本不一样.经过几天的琢磨,终于明白了真正的原理,特地写出来分享. 测试环境: $ gcc -v Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/3.2.3/specs Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --disable-checking --with-system-zlib --enable-__cxa_atexit --host=i386-redhat-linux Thread model: posix gcc version 3.2.3 20030502 (Red Hat Linux 3.2.3-24) $ gdb -v GNU gdb Red Hat Linux (6.0post-0.20031117.6rh) Copyright 2003 Free Software Foundation, Inc. GDB is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute copies of it under certain conditions. Type "show copying" to see the conditions. There is absolutely no warranty for GDB. Type "show warranty" for details. This GDB was configured as "i386-redhat-linux-gnu". $ uname -a Linux candy 2.4.21-9.EL #1 Thu Jan 8 17:03:13 EST 2004 i686 athlon i386 GNU/Linux 实例: 网上和我的这个实例雷同的也有,但是他们的是无法正确实现的.因为关键的跳转代码没有计算正确.(GCC版本问题,呵呵) /********** * a.c **********/ void function(void) { char buffer[5]. int* ret. ret=buffer 28. (*ret) =10. } void main() { int x.

x=0. function(). x=1. printf("%d\n",x). return. } /*end*/ 懂C语言的人都会认为最后的输出结果是1,可惜输出结果为0.为什么呢?请听解释. 实例分析: 相关堆栈的基础知识我就不罗嗦了,网上的介绍很多. 关键问题在于如何确定源代码 ret=buffer 28. (*ret) =10. 中的28 和 10 编译(会有warning,不用管他.) $gcc -g -o a a.c //加上-g 用来在gdb中调试 $gdb a (gdb)disas main //得到反汇编代码 如下: Dump of assembler code for function main: 0x08048366 : push 0x08048367 : mov %esp, 0x08048369 : sub $0x8,%esp 0x0804836c : and $0xfffffff0,%esp 0x0804836f : mov $0x0, 0x08048374 : sub ,%esp 0x08048376 : movl $0x0,0xfffffffc() 0x0804837d : call 0x8048348 0x08048382 : movl $0x1,0xfffffffc() 0x08048389 : sub $0x8,%esp 0x0804838c : pushl 0xfffffffc() 0x0804838f : push $0x8048474 0x08048394 : call 0x8048288 0x08048399 : add $0x10,%esp 0x0804839c : leave 0x0804839d : ret End of assembler dump. (gdb)disas function Dump of assembler code for function function: 0x08048348 : push 0x08048349 : mov %esp, 0x0804834b : sub $0x28,%esp 0x0804834e : lea 0xffffffe8(), 0x08048351 : add $0x1c, 0x08048354 : mov ,0xffffffe4() 0x08048357 : mov 0xffffffe4(), 0x0804835a : mov 0xffffffe4(), 0x0804835d : mov (), 0x0804835f : add $0xa, 0x08048362 : mov ,() 0x08048364 : leave 0x08048365 : ret End of assembler dump. 可以得知当main中执行 0x0804837d : call 0x8048348 时 会将下一条指令的地址保存在堆栈中. 即 0x08048382 我们的目的就是要想这个值修改成下一条指令的地址 0x08048389 这样就达到了屏蔽 x=1 这条语句了. 关键问题在于如何寻找保存0x08048382这个值的地址.... 100Test 下载频道开通，各类考试题目直接下载。详细请访问