java认证:数据库中安全知识介绍Oracle认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/571/2021_2022_java_E8_AE_A4_E8_AF_81_c104_571348.htm 数据库系统是在操作系统平台之上的最重要的系统软件，数据库系统的安全可以说是十分重要的。曾经有句话这样说：如果网络遍地是金钱，那么金钱就在数据库服务器中。随着无纸化业务环境的不断扩大，人们在数据库中存储着越来越多的敏感信息：银行账户、医疗记录、政府文件、军事机密等等，数据库系统就成为越来越有价值的攻击目标，因此确保数据库系统的安全也越来越重要。 作为一种大型的系统软件，数据库系统中也存在着各种各样的安全漏洞，其中危害性较大的有缓冲区溢出、堆溢出和SQL注入等。 1.缓冲区溢出 缓冲区溢出是一种很常见也很古老的安全漏洞。早在上个世纪80年代，缓冲区溢出就已经为人所知，但时至今日，大量的缓冲区溢出漏洞仍被发现。最著名的Morris蠕虫就是利用Unix系统上fingerd程序的缓冲区溢出漏洞。在Oracle 9i发布之初，Oarcle公司曾宣称他的数据库是“ unbreakable ”的，但不到几个月的时间，就暴出Oracle 9i中oracle.exe、XDB等程序存在多个缓冲区溢出漏洞。 在C语言中最常见的缓冲区就是字符数组，而操纵字符数组的函数有gets、strcpy、sprintf等。这些函数在执行字符串拷贝的过程中没有对字符串进行长度检查，这样就很容易发生超长的字符串溢出缓冲区的情况。当初这样设计是出于效率的考虑，但现在看来，这些函数的使用已成为C语言软件脆弱的一个重要因素。如果程序员没有良好的编程习惯，时刻注意函数调用过程中是否拷贝了超过缓冲区长度的字符串，

那么缓冲区溢出就不可避免。对于一个有缓冲区溢出漏洞的程序，当普通用户输入超长字符串时，通常只会使该程序崩溃。例如对于下面一小段代码： 以下是引用片段： /* vulprog */ #include int main(int argc , char * argv[]) { char buff[8]. strcpy(buff, argv[1]). } 如果用户执行 ./vulprog AAAAAAAAAAAAAAAA，在Linux上会出现段错误，因为用户输入了超长的字符串，除了填满了缓冲区，还覆盖了其他一些程序正常退出所需要的数据。为了研究这个问题，就需要了解Linux系统中进程的内存空间。 进行函数调用时系统所作的“ 序幕 ”工作就是将函数的返回地址和EBP压栈，再将ESP赋给EBP使其成为局部基指针，最后ESP减去一定的值为局部变量留出空间。这样当程序将过长的字符串拷贝到缓冲区时就会依次覆盖EBP和返回地址。当用AAAA覆盖返回地址，函数退栈时系统就将 0x41414141(A的16进制ASCII码)赋给EIP去执行，由于是一个非法的内存地址，故程序崩溃。但如果用一个实际存在的地址覆盖返回地址，那么程序就转而去执行该地址处的指令，通常黑客会在这个地址植入所谓的shellcode，由shellcode产生一个shell，如果被攻击程序设置了suid位，那么产生的shell就是root shell，黑客也就获得了系统的最高控制权，这一过程就是基本的缓冲区溢出攻击。 覆盖函数的返回地址是比较常见的攻击方式，但缓冲区溢出攻击的手法是灵活多样的，往往在编程中的一个小小纰漏就可能导致被攻击，下面简单介绍一下几种较为高级的攻击方式。 (1)通过覆盖函数指针进行攻击： 以下是引用片段： /* vulprog */ int main(int argc , char * argv[]) { void (* fp)(char *) = (void (*)(char *))&amp.puts. char buff[256]. strcpy(buff,argc[1]).

fp(argc[2]). exit(1). } 上面这个程序在执行拷贝时没有检查边界，这样用户数据就有可能覆盖函数指针fp，如果用shllcode的地址去覆盖fp，那么函数指针调用时就会去执行shellcode。 这种覆盖函数指针的方式是一种较直接的覆盖方式(因为函数指针就在缓冲区上面)，还有一种间接的覆盖方式，就是当函数指针不直接在缓冲区上面时，通过覆盖另外一个指针来覆盖函数指针，再将shellcode的地址填充函数指针。 (2)通过覆盖.dtors区地址进行攻击： 以下是引用片段： /* vulprog */ int main(int argc ,char * argv[]) { char * pbuf = malloc(strlen(argv[2]) 1). char buff[256]. strcpy(buff,argv[1]). strcpy(pbuf,argv[2]). exit(1). } 虽然这个程序没有函数指针，但在执行第二个拷贝时，可以将任意的数据拷贝到任意的地址中(这个地址由第一个拷贝指定)，这时就可以选择用 .dtors区的地址覆盖指针pbuf，在执行第二个拷贝时将shellcode的地址拷贝至.dtors区，那么在函数退出时shellcode就会被执行。 其实针对这个程序，攻击者不仅可以覆盖.dtors区的地址，还可以覆盖GOT(全局偏移表)中exit的地址，或__deregister_frame_info的地址。 从上面的这些例子可以看出，如果编程中不注意缓冲区边界的检查，就很可能导致被溢出攻击。 由于缓冲区溢出攻击的频繁爆发，迫使很多操作系统厂商推出了不可执行堆栈、更新C库函数等措施。这些措施一定程度上遏制了普通的缓冲区溢出，但道高一尺，魔高一丈，黑客们很快就将注意力转移到新的溢出攻击上，如堆溢出。从最初的溢出重要变量(如函数指针、文件指针)到dlmalloc中 malloc-free类型的堆溢出到ptmalloc中的堆溢出，层出不穷。其实不管这些手法有多高明，最终的根源只有一个：利用程序中未对缓冲区边界进行

有效检查。 2.SQL注入 数据库系统除了可能受到缓冲区溢出的攻击外，近几年又出现了SQL注入的攻击方式，这种攻击方式被称为“ SYSDBA的恶梦 ”。SQL注入可能导致数据库系统中的普通用户窃取机密数据(如获得SYSDBA密码)、进行权限提升(如获得SYSDBA特权)等，而这种攻击方式又不需要太多计算机方面的知识，一般只要能熟练使用SQL语言即可，因此对数据库的安全构成了很大的威胁。 SQL注入的攻击方式比较简单，一般是将一些特权语句注入到有漏洞的存储过程或触发器中导致这些语句被非法执行。例如在Oracle中由SYS创建如下存储过程并将执行权限授予普通用户： 以下是引用片段： CREATE OR REPLACE PROCEDURE PROC1 (INPUT VARCHAR2) AS ... ... STMT:= ’ SELECT TITLES FROM BOOKS WHERE AUTHOR = ’ ’ ’ || INPUT || ’ ’ ’ ’. EXECUTE IMMEDIATE STMT. ... ... 正常情况下用户可以通过执行：EXEC SYS.PROC1(’ DICKENS ’)来查询DICKENS的著作，但如果恶意用户这样执行该存储过程： EXEC SYS.PROC1(’ DICKENS ’ ’ UNION SELECT PASSWORD FROM USERS_TABLE WHERE ’ ’ A ’ ’ = ’ ’ A ’)，那么他就非法地查出了所有用户的密码。 虽然这只是一个简单的例子，但它表明在编写系统存储过程、函数和触发器时一定要注意防止SQL注入的可能。 数据库是信息系统的基石，一旦被黑客入侵，后果将不堪设想。而抵抗黑客入侵的最好办法就是克服软件编程中存在的各种漏洞，让黑客无机可乘。通过源码审计、漏洞跟踪等方式可以较好的修正现存系统中的各种安全隐患。目前我们正在达梦数据库中积极开展漏洞发掘的相关工作，努力使达梦数据库成为真正牢不可破的数

据库，为国家的信息安全构筑坚强的基石。 更多优质资料尽在百考试题论坛 百考试题在线题库 java认证更多详细资料 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com