

第2章 黑客常用的攻击方法



学习目标

2.8

缓冲区溢出

2.8 缓冲区溢出 (buffer overflow)

从一个对话框说起.....





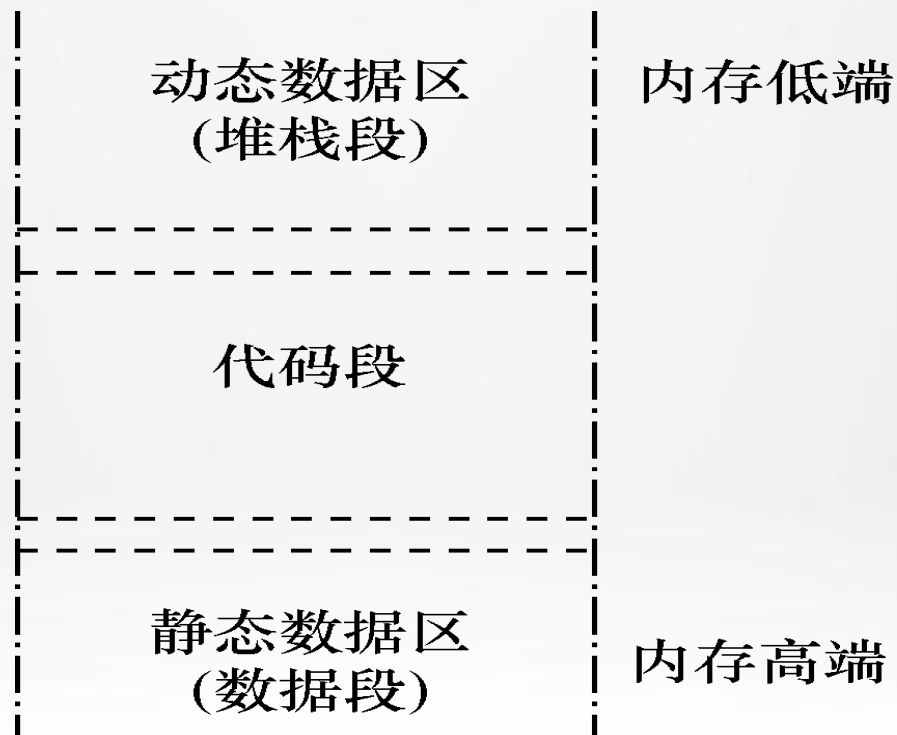
认识缓冲区溢出

引例：把1升的水注入容量为0.5升的容量中

- 第一次大规模的缓冲区溢出攻击是发生在1988年的Morris蠕虫，它造成了6000多台机器被瘫痪，损失在\$100 000至\$10 000 000之间，利用的攻击方法之一就是fingerd的缓冲区溢出。
 - 缓冲区溢出攻击已经占了网络攻击的绝大多数，据统计，大约80%的安全事件与缓冲区溢出攻击有关。
-



缓冲区溢出原理



2.7.3 Windows 系统的内存结构

计算机运行时，系统将内存划分为3个段，分别是代码段、数据段和堆栈段。

■ 代码段

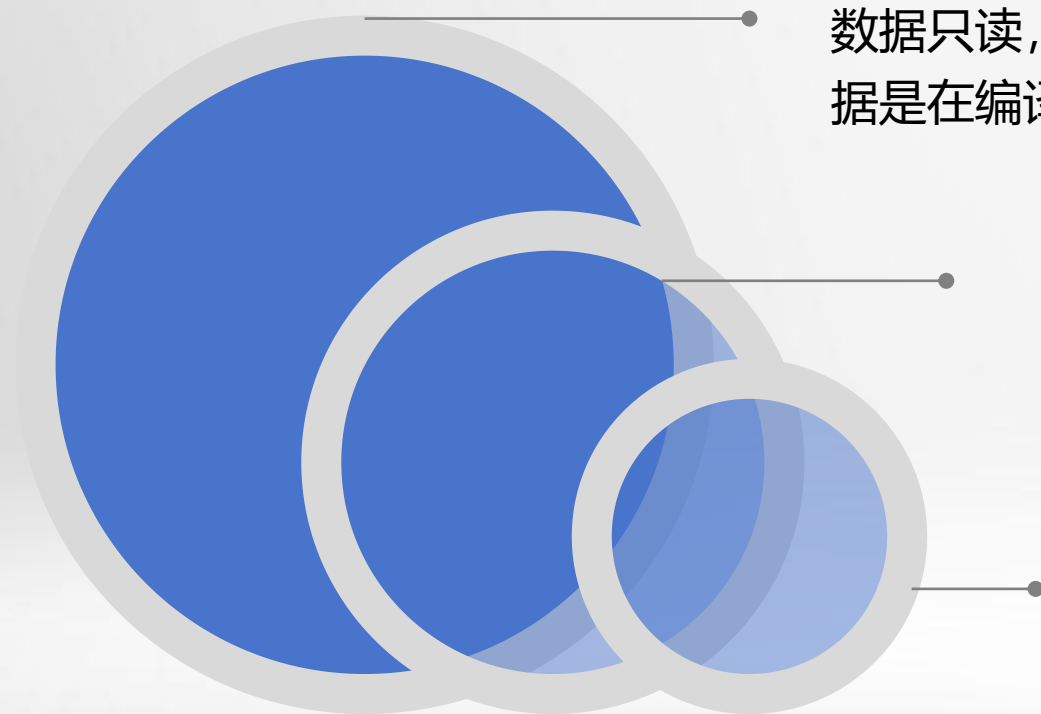
数据只读，可执行。在代码段一切数据不允许更改。在代码段中的数据是在编译时生成的2进制机器代码，可供CPU执行

■ 数据段

静态全局变量是位于数据段并且在程序开始运行的时候被加载，可读、写。

■ 堆栈段

放置程序运行时动态的局部变量，即局部变量的空间被分配在堆栈里面，可读、写。





缓冲区溢出的基本原理

- 缓冲区溢出源于程序执行时需要存放数据的空间，也即我们所说的缓冲区。
- 缓冲区的大小是程序执行时固定申请的。然而，某些时候，在缓冲区内装载的数据大小是用户输入的数据决定的。程序开发人员偶尔疏忽了对用户输入的这些数据作长度检查，由于用户非法操作或者错误操作，输入的数据占满了缓冲区的所有空间，且超越了缓冲区边界延伸到缓冲区以外的空间。我们称这个动作为缓冲区溢出。





缓冲区溢出的基本原理

- 缓冲区溢出是由于系统和软件本身存在脆弱点所导致的。
 - 例如目前被广泛使用的C和C++，这些语言在编译的时候没有做内存检查，即数组的边界检查和指针的引用检查，也就是开发人员必须做这些检查，可是这些事情往往被开发人员忽略了；标准C库中还存在许多不安全的字符串操作函数，包括：strcpy(), sprintf(), gets()等等，从而带来了许多脆弱点，这些脆弱点也便成了缓冲区溢出漏洞。
-

2.7.3 Windows 系统的内存结构



Windows缓冲区溢出实例分析

```
/*  
* 文件名 : overflow.cpp  
* 功能 : 演示Windows缓冲区溢出的机制  
*/  
  
#include <stdio.h>  
#include <string.h>  
  
char bigbuff[]="aaaaaaaaaa"; // 10个a  
  
void main()  
{  
    char smallbuff[5];          // 只分配了5字节的空间  
    strcpy(smallbuff,bigbuff);  
}
```

2.7.3 Windows 系统的内存结构

OllyDbg - overflow.exe - [CPU - 主线程, 模块 - overflow]

文件(F) 查看(V) 调试(D) 插件(P) 选项(O) 窗口(W) 帮助(H)

地址 HEX 数据 反汇编 寄存器 (FPU)


地址	HEX 数据	反汇编	寄存器 (FPU)
00401000	\$ 51	PUSH ECX	EAX 0012FF7C
00401001	51	PUSH ECX	ECX 00370768
00401002	8D4424 00	LEA EAX, DWORD PTR SS:[ESP]	EDX 00370000
00401006	68 30604000	PUSH overflow.00406030	EBX 7FFDF000
0040100B	50	PUSH EAX	ESP 0012FF74
0040100C	E8 0F000000	CALL overflow.00401020	EBP 0012FFC0
00401011	83C4 10	ADD ESP, 10	ESI 00000000
00401014	C3	RETN	EDI 00000000
00401015	CC	INT3	EIP 0040100C overflow.0040100C
00401016	CC	INT3	C 0 ES 0023 32位 0 (FFFFFFFF)
00401017	CC	INT3	P 1 CS 001B 32位 0 (FFFFFFFF)
00401018	CC	INT3	A 0 SS 0023 32位 0 (FFFFFFFF)
00401019	CC	INT3	Z 0 DS 0023 32位 0 (FFFFFFFF)
0040101A	CC	INT3	S 0 FS 003B 32位 7FFDE000 (FFF)
0040101B	CC	INT3	T 0 GS 0000 NULL
0040101C	CC	INT3	D 0
0040101D	CC	INT3	O 0 LastErr ERROR_SUCCESS (0000)
0040101E	CC	INT3	EFL 00000206 (NO, NB, NE, A, NS, PE, 0)
0040101F	CC	INT3	ST0 empty 0.0
00401020	\$ 57	PUSH EDI	ST1 empty 0.0
00401021	8B7C24 08	MOV EDI, DWORD PTR SS:[ESP+8]	ST2 empty 0.0
00401025	EB 6A	JMP SHORT overflow.00401091	ST3 empty 0.0
00401027	8D	DB 8D	ST4 empty 0.0
00401028	8D	DB 8D	ST5 empty 0.0
00401029	8D	DB 8D	ST6 empty 0.0

本地调用来自 <模块入口点>+0AF

地址	HEX 数据	0012FF74	0012FF7C	0012FF78	0012FF7C	0012FF78	0012FF7C
00406000	00 00 00 00 00 00 00 00 00 00 00 00 2F 25	0012FF78	00406030	ASCII "aaaaaaa"	0012FF7C	00370768	00370768
00406010	00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FF80	00370768	0012FF84	004011C4	返回到 overflow	0012FF88
00406020	00 00 00 00 00 00 00 00 00 00 00 00 00 00	0012FF88	00000001	0012FF8C	00370C60	0012FF90	00370CA8
00406030	61 61 61 61 61 61 61 61 61 61 61 61 61 61						
00406040	78 12 40 00 01 00 00 00 05 00 00 C0 0B 00						
00406050	00 00 00 00 1D 00 00 C0 04 00 00 00 00 00						
00406060	96 00 00 C0 04 00 00 00 00 00 00 00 8D 00						

命令 : 暂停

2.7.3 Windows 系统的内存结构

 调用strcpy()函数时堆栈的填充情况



2.7.3 Windows 系统的内存结构



执行strcpy()函数的过程

0012FF6C	00000000	
0012FF70	00401011	返回到 overflow.00401011 来自
0012FF74	0012FF7C	
0012FF78	00406030	ASCII "aaaaaaaaaa"
0012FF7C	00370768	
0012FF80	00370768	
0012FF84	004011C4	返回到 overflow. <模块入口点>
0012FF88	00000001	

(1)

0012FF6C	00000000	
0012FF70	00401011	返回到 overflow.00401011 来自
0012FF74	0012FF7C	
0012FF78	00406030	ASCII "aaaaaaaaaa"
0012FF7C	61616161	
0012FF80	00370768	
0012FF84	004011C4	返回到 overflow. <模块入口点>
0012FF88	00000001	

(2)

0012FF6C	00000000	
0012FF70	00401011	返回到 overflow.00401011 来自
0012FF74	0012FF7C	
0012FF78	00406030	ASCII "aaaaaaaaaa"
0012FF7C	61616161	
0012FF80	61616161	
0012FF84	004011C4	返回到 overflow. <模块入口点>
0012FF88	00000001	

(3)

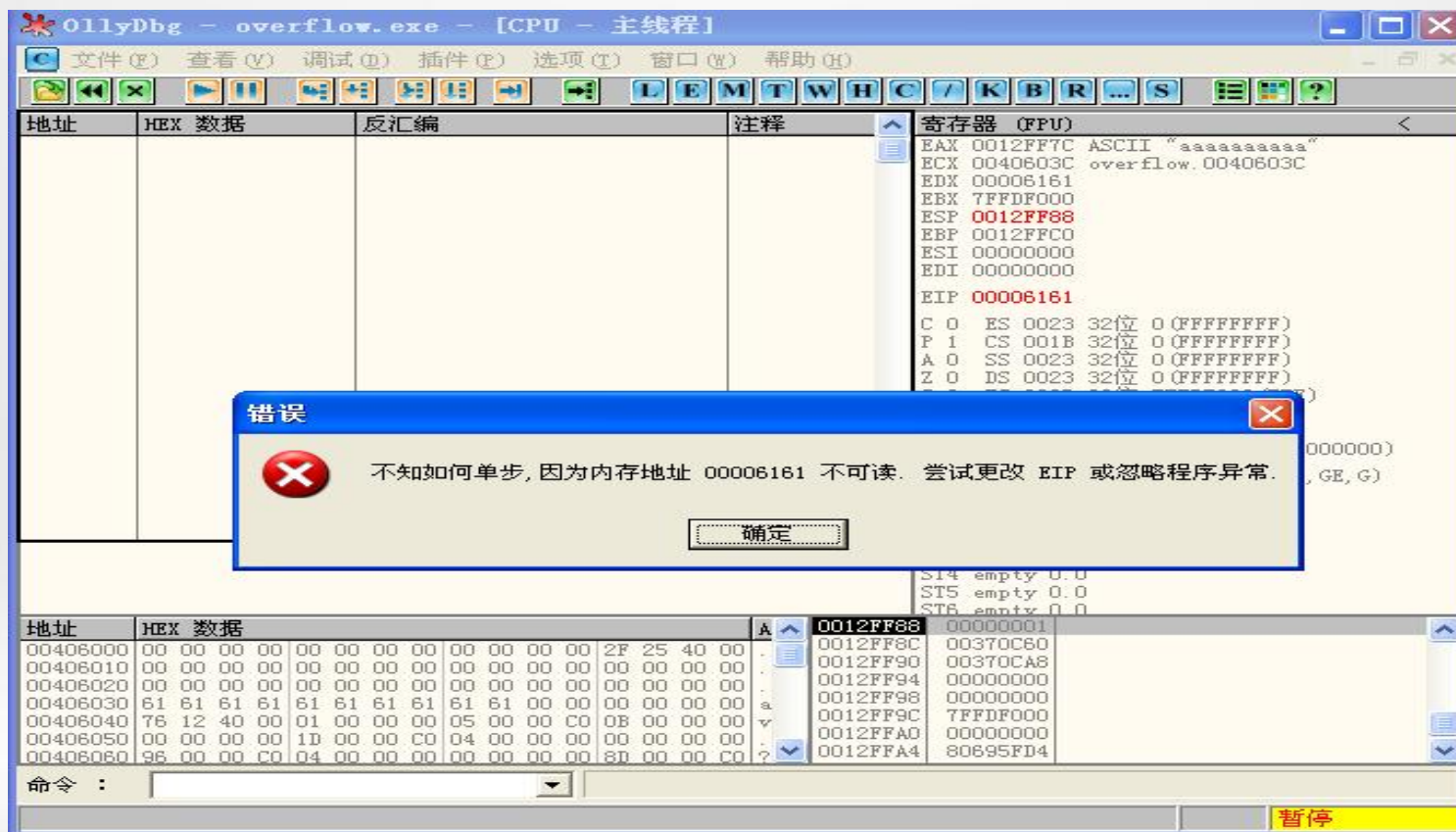
0012FF6C	00000000	
0012FF70	00401011	返回到 overflow.00401011 来自
0012FF74	0012FF7C	ASCII "aaaaaaaaaa@"
0012FF78	00406030	ASCII "aaaaaaaaaa"
0012FF7C	61616161	
0012FF80	61616161	
0012FF84	00406161	overflow.00406161
0012FF88	00000001	

(4)

2.7.3 Windows 系统的内存结构



溢出结果



2.7.3 Windows 系统的内存结构

缓冲区溢出的危害

可以导致程序运行失败、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。而缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，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。





缓冲区溢出攻击的实验分析

2000年1月，Cerberus 安全小组发布了微软的IIS 4/5存在的一个缓冲区溢出漏洞。攻击该漏洞，可以使Web服务器崩溃，甚至获取超级权限执行任意的代码。目前，微软的IIS 4/5 是一种主流的Web服务器程序；因而，该缓冲区溢出漏洞对于网站的安全构成了极大的威胁。它的描述如下：

- 浏览器向IIS提出一个HTTP请求，在域名（或IP地址）后，加上一个文件名，该文件名以 “.htr” 做后缀。于是IIS认为客户端正在请求一个 “.htr” 文件， “.htr” 扩展文件被映像成ISAPI（Internet Service API）应用程序，IIS会复位向所有针对 “.htr” 资源的请求到 ISM.DLL程序，ISM.DLL 打开这个文件并执行之。
- 浏览器提交的请求中包含的文件名存储在局部变量缓冲区中，若它很长，超过600个字符时，会导致局部变量缓冲区溢出，覆盖返回地址空间，使IIS崩溃。

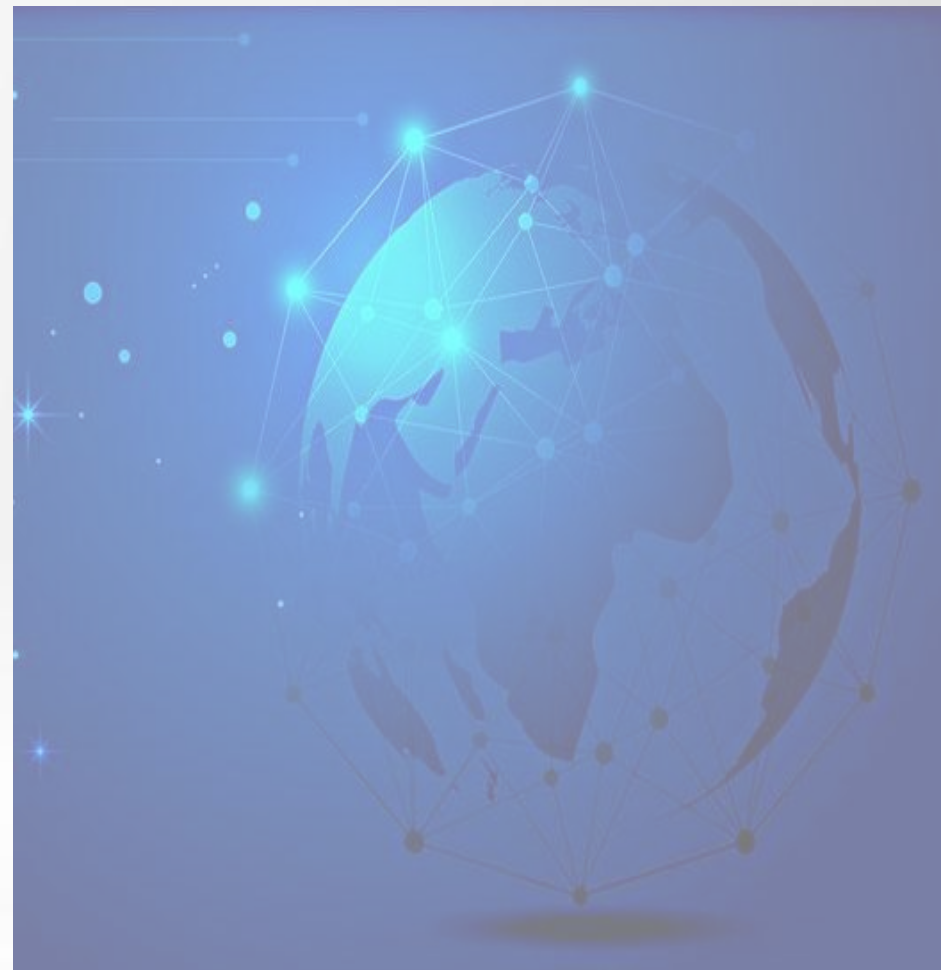


缓冲区溢出攻击

上述的缓冲区溢出例子中，只是出现了一般的拒绝服务的效果。但是，实际情况往往并不是这么简单。当黑客精心设计这一EIP，使得程序发生溢出之后改变正常流程，转而去执行他们设计好的一段代码

（也即ShellCode），攻击者就能获取对系统的控制，利用ShellCode实现各种功能，比如，监听一个端口，添加一个用户，等等。这也正是缓冲区溢出攻击的基本原理。

目前流行的缓冲区溢出病毒，如冲击波蠕虫、震荡波蠕虫等，就都是采用同样的缓冲区溢出攻击方法对用户的计算机进行攻击的。



2.7.3 Windows 系统的内存结构



流行的缓冲区溢出攻击病毒

冲击波

利用漏洞：RPC缓冲区溢出 135/TCP

1

震荡波

利用漏洞：LSASS漏洞 1025/TCP

2

高波

利用多种漏洞，非常危险

4

极速波

利用漏洞：UPNP漏洞 445/TCP

3



防范缓冲区溢出攻击的有效措施



强制程序开发人员书写正确的、安全的代码

目前，可以借助grep、FaultInjection、PurifyPlus等工具帮助开发人员发现程序中的安全漏洞。



通过对数组的读写操作进行边界检查来实现缓冲区的保护，使得缓冲区溢出不可能出现，从而完全消除了缓冲区溢出的威胁

常见的对数组操作进行检查的工具具有Compaq C编译器，Richard Jones和Paul Kelly开发的gcc补丁等。



微软的DEP（数据执行保护）技术



通过操作系统设置缓冲区的堆栈段为不可执行，从而阻止攻击者向其中植入攻击代码

微软的DEP（数据执行保护）技术
（Windows XP SP2、Windows Server 2003 SP1及其更高版本的Windows操作系统中）

