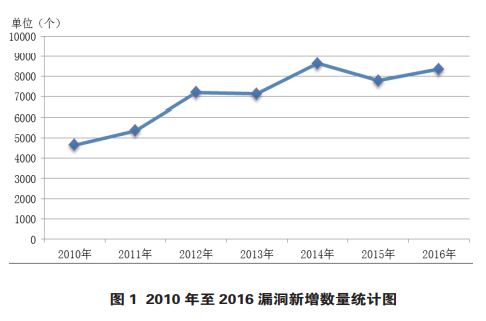
多平台Shellcode开发与实现原理研究

#### 第一章 绪论

##### 1.1 研究背景

近年随着计算机技术飞速发展，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》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》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。

##### 1.2 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。

##### 1.3 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。
3. 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。
4. 开发具有远程下载功能的shellcode实例
5. 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

#### 第二章 SHELLCODE概述

##### 2.1 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。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。

##### 2.2 SHELLCODE 基本类型

Shellcode可以分为本地shellcode和远程shellcode，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。

#### 第三章 基于windows平台的SHELLCODE开发

##### 3.1 开发流程

Windows平台下shellcode通过调用系统API实现预期功能，具体流程包括：

1、定位kernel32.dll 基址

1. 定位LoadLibrary函数地址
2. 通过LoadLibrary加载DLL并获得其基址
3. 获取被加载的DLL中的函数地址
4. 通过CALL指令调用该函数
5. 通过特定函数实现shellcode预定功能

##### 3.2 关键技术

###### 3.2.1 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]

1. 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。

1. 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）。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。

###### 3.2.3 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。

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。

###### 3.3.3 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。

##### 3.3 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。

在kernal32.dll定位方法方面：

1. 由FS寄存器中存储TEB的GDT序号改为在GS寄存器中存储。
2. 各数据结构的偏移量有所不同

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，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。
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。

#### **第四章 基于linux平台的SHELLCODE开发**

##### 4.1 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。

##### 4.2 关键技术

###### 4.2.1 借助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

1. 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

1. 将//sh入栈；

push 0x68732f2f

1. 将/bin入栈；

push 0x6e69622f

1. 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。

###### 4.2.2 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。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

##### 4.3 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，其中主要使用rax、rdi、rsi和rdx来存储系统调用号和其他参数。
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，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。

#### **第五章 具有远程下载功能的多平台shellcode实现（metasploit）**

##### 编写shellcode一般经过以下3个步骤： 1.编写简洁的能完成所需要功能的c程序； 2.反汇编可执行代码，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.提取出操作码，写成shellcode，并用C程序验证

##### 5.1 WIN32平台

##### 5.2 WIN64 平台

##### 5.3 LINUX\_X86平台

##### 5.4 LINUX\_X86-64平台

#### 第六章 总结与展望

#### **参考文献**