一种抗地址淹没的缓冲区栈溢出算法*

唐成华[1,2]，彭 灿[1]，刘 猛[1]，钱俊彦[1,2]
(1. 桂林电子科技大学 广西可信软件重点实验室，桂林 541004；2. 广西密码学与信息安全重点实验室，广西桂林 541004)

摘 要：缓冲区溢出是常见的网络漏洞攻击，其中最重要的是栈溢出攻击。通过分析缓冲区溢出攻击的方法和特点，提出一种基于 StackShield 改进的 RetProtect 算法，首先利用 IDA Pro 对源程序反汇编分析，然后建立新的库函数，并通过修改 GCC 源代码来实现程序执行时对函数返回地址的备份的方法来检测缓冲区溢出攻击的发生。与其他栈溢出攻击检测方法相比，RetProtect 算法可有效地阻止对返回地址进行淹没的栈溢出攻击，对用户透明，系统兼容性好。

关键词：缓冲区溢出；栈溢出；地址淹没；返回地址备份
中图分类号：TP309 **文献标志码**：A **文章编号**：1001-3695(2017)12-3758-04
doi:10.3969/j.issn.1001-3695.2017.12.054

Anti-address flooding algorithm for buffer stack overflow

Tang Chenghua[1,2], Peng Can[1], Liu Meng[1], Qian Junyan[1,2]
(1. *Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin* 541004, *China*; 2. *Guangxi Key Laboratory of Cryptography & Information Security, Guilin* 541004, *China*)

Abstract: Buffer overflow is common network vulnerability, and the most important one is the stack overflow attack. By analyzing the methods and characteristics of buffer overflow attacks, this paper proposed an improved RetProtect algorithm based on StackShield. It used IDA Pro for the disassembly analysis of the source program, and then established a new library function. It detected the occurrence of buffer overflow attacks by modifying the GCC source code to realize the backup of the function return address when the program executed. Compared with other stack overflow attack detection methods. The RetProtect algorithm can effectively prevent the stack overflow attacks on the return address overlay, which is transparent to the user and good compatibility.

Key words: buffer overflow; stack overflow; address flooding; return address backup

0 引言

人们生活和工作已与互联网越来越紧密。随着各种应用软件，尤其是手机 APP 的广泛应用，安全问题也日益突出。例如盗取银行卡和游戏账号、公司内网入侵、勒索软件、验证码诈骗短信等，而棱镜门和斯诺登事件的发生更使得网络安全问题第一次全面曝光在大众眼前，人们已认识到安全问题在全球范围内相当严峻。

当前网络安全的威胁主要来自于黑客，在其使用的攻击方式中，又以利用缓冲区溢出漏洞最为常见。作为普遍存在于各种软件和操作系统中的一种漏洞，缓冲区溢出很容易被攻击者利用，这是因为诸如 C/C ++ 等没有自动检测缓冲区溢出的指令，而通过编写代码来始终检查缓冲区是否可能溢出的时间成本太大，所以通常是在某个缓冲区溢出漏洞被曝光时通过发布补丁来修复漏洞。据中国国家信息安全漏洞库(CNNVD)[1]发布的最新的《信息安全漏洞态势报告(2015 年度)》统计，在 2015 年所有新增漏洞中，数量最多的是各种缓冲区溢出漏洞，高达 1 088 个，占比高达 14.03%，与 2014 年该类型漏洞数量 787 个相比，增加了 38.25%。缓冲区溢出漏洞造成的危害范围和程度都非常大，利用缓冲区溢出漏洞，可以实现如堆栈摧毁、未授权指令获取、木马上传、任意代码执行[2]等系统攻击。如何防范缓冲区溢出漏洞攻击一直是漏洞分析研究领域的热点。

1 相关技术及分析

最早在 1996 年，Aleph One[3]首次详细描述了 Linux 系统中栈的结构，并提出了如何利用基于栈的缓冲区溢出向进程中植入一段用于获得 shell 的代码(shellcode)。基于这种溢出思想，攻击者可以在漏洞程序内存的任意位置加入恶意代码，导致操作系统崩溃从而拒绝服务，甚至是操纵程序执行流程中的关键数据(如函数调用后的返回地址)，从而控制程序的执行流程，取得与目标主机上的相关程序一致的 shell 控制权并实

收稿日期：2016-11-09；**修回日期**：2017-01-03 **基金项目**：国家自然科学基金资助项目(61462020,61562015)；广西自然科学基金资助项目(2014GXNSFAA118375)；广西可信软件重点实验室项目(kx201506)；广西密码学与信息安全重点实验室课题(GCIS201619)；广西高等学校高水平创新团队及卓越学者计划资助

作者简介：唐成华(1974-)，男，湖北黄冈人，副教授，博士(后)，主要研究方向为网络信息安全(tch@guet.edu.cn)；彭灿(1991-)，男，安徽芜湖人，硕士研究生，主要研究方向为网络信息安全；刘猛(1992-)，男，江苏新沂人，硕士研究生，主要研究方向为网络信息安全；钱俊彦(1973-)，男，浙江嵊县人，教授，博士，主要研究方向为软件形式化安全分析.

施攻击。如果被利用的程序以 root 权限运行,则攻击者就能够通过 shell 得到该权限从而获得对主机的完全控制。

缓冲区溢出攻击方法以格式化字符串漏洞(format string vulnerability)、栈溢出(stack smashing)、堆溢出(heap smashing)和整数溢出(integer variable overflow)等为常见,尤其是在 Linux 系统中[4]。通过缓冲区溢出漏洞现状分析可知,针对栈溢出的攻击仍然是主流形式。因为栈溢出攻击实现相对简单,一旦溢出成功后又对系统的破坏性很大,例如作为 2015 年开源软件的十大安全漏洞之一,在 Linux Kernel KCodesNetUSB 模块中的 run_init_sbus 函数存在基于栈的缓冲区溢出漏洞,因为服务端缺乏足够的验证,远程攻击者可通过客户端在 TCP 20005 端口的会话中提供大于 64 字符长度的计算机名,使得那些含有 NetUSB 模块的设备出现内核溢出,从而造成内存破坏,并可能演变成远程代码攻击或者 DoS 攻击。Linux kenel Kcodes NetUSB 模块存在于数以百万计的多款 Netgear 和 TP-Link 路由和嵌入式设备产品中,攻击者通过利用该漏洞,可以在内核层面远程执行恶意代码,控制路由等网络设备,受到了国内外广泛关注。而格式化字符串攻击、堆溢出攻击等执行成功的难度较大,特别是微软在 VS2008 之后就全面引入安全函数机制,对 C/C ++ 不安全函数进行标记,一旦检测到程序中使用了这些函数就会报错。

针对缓冲区溢出攻击,一些学者和研究人员也提出了各种防御方法。如数组边界和指针检查技术[5]、通过新建一个 C 库来代替标准 C 库的 Libsafe[6] 栈区保护、针对函数返回地址进行检测的 StackGuard[7,8] 技术、栈溢出保护(SSP)[9] 技术、单字节栈溢出分析[10],以及与 StackGuard 类似的 StackShield[11] 等。但一些问题仍然存在,如仅以数组边界和指针检查是需要先获得程序的源代码,而且还会产生大量的误判(false positive 或 true negative)。Libsafe 则是只保护栈区,不考虑 heap 和 bss 区,所以攻击者仍可通过对 heap 和 bss 里的某些重要变量进行覆盖,利用 shellcode 获得系统控制权。当程序被编译时,StackGuard 在返回地址和变量之间放置一个探针(canary)来检测返回地址是否被修改或覆盖,从而判断是否有溢出攻击,这种每次执行都需要对探针值进行检查,导致执行效率差,而且在重新编译程序的时候需要程序的源代码。作为一个 GNU C 编译器的扩展,StackShield 也是通过对函数返回地址进行保护来阻止缓冲区溢出攻击,不过在保护函数的返回地址被更改的方面很有效,在函数被调用时,它将函数的返回地址保存到一块未写入任何数据的内存中,如果返回时检测返回地址和原先保存的地址不一致(堆栈上的函数返回地址被修改),则用原先保存的地址来代替函数返回地址,这样确保函数正确返回。显然,StackShield 存在编译阶段重汇编的效率问题,以及由于 Linux 系统中存在不同平台的不同语法和汇编码,而导致处理编译生成的汇编语言文件时可能会出现各种错误的兼容性问题。

2　基于地址淹没的缓冲区溢出攻击

通常,栈溢出攻击是对被调用函数的返回地址进行覆盖或淹没达到执行恶意跳转指令而溢出的目的,这在缓冲区溢出攻击中更为方便,也成为溢出防御研究领域围绕怎样保护函数的返回地址不被修改的动因。

在程序执行的反汇编代码过程中,在每个函数被调用之前,都先将各自参数压入堆栈中,然后执行 call 指令来调用该函数,同时将 call 指令的下一语句(即返回地址)压入堆栈,并保存原先的基址指针 EBP,提升栈底,建立框架指针,并提升堆栈,为函数的局部变量预留一定的空间;其次,保留现场,将函数执行中可能用到的一些寄存器也压入栈中,然后利用 lea 和 int 3 指令将前面分配的缓冲区全部填充为 cc,这样即使出现问题也会由于 int 3 指令而终止操作;接着,实现该函数本身的功能;最后,恢复之前保存的寄存器值,恢复栈底,执行 retn 指令返回,函数调用结束。

一个缓冲区溢出漏洞的 C 语言程序如下:

```
fun()
{int i;
int a[5] = {0};
for i = 0; i≤5; i ++)
        {a[i] = 4;
        printf("hello world! \n");
        }
}
main(int argc,char * argv[])
{fun();
return 0;
}
```

这里给出的 fun 函数首先为数组分配了五个元素,但是在 for 循环中却使用了 a[5]而造成数组越界,即导致缓冲区溢出,使得程序循环执行打印“hello world!”的操作而无法正常返回。或者说,在其反汇编代码中,ret 指令的下一条指令即返回地址被修改(被 a[5]淹没),从而改变了整个程序的执行流程。

3　RetProtect 算法

在逆向工程中,交互式反汇编器专业版 IDA Pro 可对程序进行静态反汇编分析,以强大的标注功能著称,能够自动标注 VC、Borland C、Delphi、C/C ++ 等常见编译器中的标准库函数,借助 IDA Pro 可确定程序中各函数的起始地址和大小,并描绘出函数调用图,这为本文确定程序中缓冲区的大小提供了一个思路。IDA Pro 的函数识别如图 1 所示。

Function name	Segment	Start	Length
sub_401000	.text	00401000	000000BE
main	**.text**	**004010C0**	**00000024**
sub_4010F0	.text	004010F0	00000031
_scanf	**.text**	**00401121**	**00000017**
start	.text	00401138	000000DF
__amsg_exit	.text	00401217	00000022
_fast_error_exit	.text	0040123C	00000023
__stbuf	.text	00401260	0000008D
__ftbuf	.text	004012ED	0000003D
sub_40132A	.text	0040132A	0000077E
_write_char	.text	00401AC8	00000035
_write_multi_char	.text	00401AFD	00000031
_write_string	.text	00401B2E	00000038
_get_int_arg	.text	00401B66	0000000D
_get_int64_arg	.text	00401B73	00000010
_get_short_arg	.text	00401B83	0000000E
__input	.text	00401C4A	00000A25
__hextodec	.text	0040266F	00000037
_fgetc	**.text**	**004026A6**	**0000001A**

图 1　IDA Pro 函数识别图

基于以上思路和相关方法,本文研究了一种基于 StackShield 改进的 RetProtect 算法,其主要思路如下:

a)在程序执行时,操作系统会为执行过程分配相应的缓冲区空间,此时通过 IDA Pro 对程序进行静态分析以确定程序中相关函数名称、地址和大小;b)在缓冲区中新建一个数组用于存放函数的返回地址,每调用一个函数就将该函数的返回地址添加到该数组中去;c)每当函数执行完返回时,判断函数的返回地址和数组中保存的函数地址是否一致,即返回地址是否被非法修改,确定该函数是否存在潜在的缓冲区溢出漏洞,以

及程序是否报错。算法流程如图 2 所示。

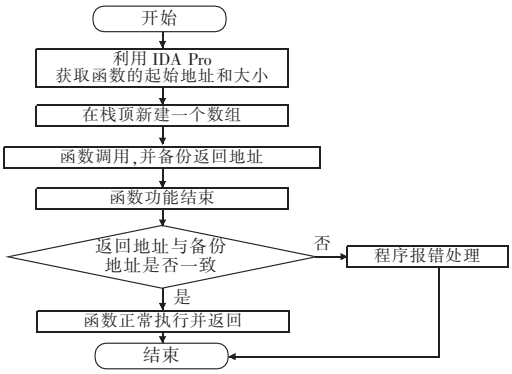

图 2 RetProtect 算法流程

针对 StackShield 存在的执行效率差和兼容性不好的问题,通过编写库函数文件,并修改 GCC 源代码,实现程序执行时对函数返回地址的备份的方法来解决。具体如下:

首先在库函数里声明一个数组,在函数的入口处调用入口函数,把此时程序堆栈中的返回地址备份到数组中去。当函数正常执行完准备返回时,在函数的结束部分,比较先前数组中保存的返回地址和当前函数的返回地址是否一致:如果一致,则认为程序执行正常,否则认为函数返回地址可能被人恶意修改过,可能会存在缓冲区溢出的情况。

其中库函数 retarray. c 的实现如下:

```
#include〈fcntl. h〉
#include〈unistd. h〉
#include〈string. h〉
#include〈stdio. h〉
#define LENGTH 256
#define_PATH_TTY "/dev/tty"
/ * Use an array to save the return address for each invoked function * /
void * _retarray[LENGTH];
int_retindex = 0;
void _retarray_begin(void * cur_addr)
{if (_retindex < LENGTH)
{_retarray[_retindex] = cur_addr;
__retindex ++;
return;
}
void _retarray_end(void * cur_addr)
{int fd;
fd = open(_path_TTY,O_WRONLY);
_retindex --;
if(_retindex < LENGTH)
{if (cur_addr! = __retarray[_retindex])
{write(fd," +++ different return address, the program
will be terminated! +++ \n",33);
close(fd);
_exit(127);
}
}
return;
close(fd);
}
```

在库函数 retarray. c 中,_retarray 是在堆栈中新建的用于备份函数返回地址的数组,_retindex 是该数组的索引,数组的长度设置为 256(通常,程序的递归调用深度不超过 256 次,否则程序会崩溃)。_retarray_begin 函数和_retarray_end 函数分别是 GCC 编译器在编译每个函数的时候在其开始部分和结束部分调用的函数。cur_addr 则是编译器从当前数组中取出的已保存的函数返回地址。

另外,还对 GCC 的源码进行修改,以便使得 GCC 可以正常编译通过,有以下几步:

a)增加调试选项。为了方便,直接在 GCC 中防止缓冲区溢出的选项的基础上增加子选项-fstack-ret-protect,这需要对 GCC 目录下的 common. opt 文件进行修改,具体的,增加了如下几行:

```
fstack-ret-protect
Common Report RejectNegative Var(flag_stack_protect,3)
Use an array to save the return address for each invoked function
```

当使用 GCC 使用-fstack-ret-protect 选项时,GCC 源码中变量 flag_stack_protect 的值将被设置为 3。

b)在 gcc/function. c 中的 expand_function_start 函数和 expand_function_end 函数中,增加 ret_addr_backup_prologue( ) 和 ret_addr_backup_epilogue( ) 两个函数调用,主要用于生成对_retarray_begin 和_rtrarray_end 的调用。其对应的代码段如下:

```
if strcmp(current_function_name( ),"main") &&(flag_stack_protect
==3) & (cfun->calls_alloca))
ret_addr_backup_prologue( );
if(strcmp(current_function_name( ),"main") &&(flag_stack_protect
==3) && (cfun->calls_alloca))
ret_addr_backup_epilogue( );
static void ret_addr_backup_prologue(void)
{rtx pro_func,addr_rtx;
pro_func = gen_rtx_SYMBOL_REF(Pmode,"_retarray_prolog");
addr_rtx = gen_rtx_MEM(Pmode,plus_constant(gen_rtx_MEM
(Pmode,plus_constant (stack_pointer_rtx,8)),0));
fp_rtx = gen_rtx_MEM(Pmode,plus_constant(stack_pointer_rtx,8));
emit_library_call(pro_func,LCT_NORMAL,VOIDmode,1,addr_rtx,
Pmode);
return;
}
```

以上代码中未给出_retarray_epilogue 函数的实现,其实现方法类似,此处略。

4 实验过程与分析

4.1 实验环境

为了验证 RetProtect 算法的有效性,进行了相关实验和性能分析。实验硬件平台环境基于虚拟机 Xubuntu14. 04. 4 LTS (32 位)系统,内核版本号为 4. 2. 0-27-generic,内存大小为 1 GB,GCC 版本为 4. 5. 3。

4.2 实验方法

借助 RIPE[12] 工具生成各种溢出攻击,并综合评估算法防缓冲区溢出的能力。该工具软件能从攻击位置、攻击目标、攻击模式、攻击手段和攻击函数五种攻击向量来对防缓冲区溢出能力进行评价。由于 RetProtect 算法主要是针对栈溢出函数的返回地址进行防御的,所以本文的实验仅考虑后四种攻击向量。

实验过程首先是在 Xubuntu 系统下,通过对存在缓冲区溢出漏洞的实验程序进行编译运行,通过对输出的结果,判断能否检测出全部的缓冲区溢出攻击。其次是进一步通过模拟攻击来验证防御方法的有效性。

4.3 结果分析

StackGuard、ProPlice、StackShield 和 RetProtect 算法在四种攻击向量上的防御能力情况如表 1 所示。

表 1　不同算法的防御能力

攻击向量	StackGuard	ProPolice	StackShield	RetProtect
攻击目标	返回地址	返回地址栈指针	返回地址	返回地址
攻击模式	直接攻击	直接攻击	直接攻击 间接攻击	直接攻击 间接攻击
攻击手段	不区分	不区分	不区分	不区分
攻击函数	不区分	不区分	不区分	不区分

基于四个维度的防御能力的比较,从表 1 可以看出,ProPlice 和 StackGuard 是采用类似于探针的保护方法,因此在攻击模式上,这种探针方法只能处理直接攻击而不能处理间接攻击。另一方面,由于 RetProtect 算法是基于 StackShield 方法的改进,它们在防御能力有效性的方面保持一致,但是通过大量用例程序的测试可以发现 RetProtect 算法比 StackShield 算法在防御效果方面更好,如表 2 所示。

表 2　不同算法的防御有效性

防御方法	成功数/个	失败数/个	成功率/%
Xubuntu14.04	0	850	0
gcc-fstack—protector	81	769	9
gcc-fstack-protector-all	92	758	11
Proplice	354	496	42
Stackshield	310	540	36
RetProtect	380	470	45

表 2 是有效性验证实验结果,其中 gcc-fstack-protector 和 gcc-fstack-protector-all 是 GCC 自带的检测缓冲区溢出的选项,可以看出 RetProtect 算法相比其他算法效果更好。

借鉴于文献[13]的性能评估方法,各缓冲区溢出防御方法相对于未采取防御措施时的函数调用的性能开销数据对比如表 3 所示。其中 ProPolice 利用 GCC 自带的安全检查选项-fstack-protector 等来实现,而 StackShield 则通过 Global Ret Stack 来实现。从实验结果可以看出,RetProtect 方法和其余方法相比性能开销不大,在可接受的合理范围内。

表 3　不同算法的的系统时钟开销/CPU 时钟周期数

测试函数	未保护	ProPolice	StackSheild	RetProtect
fun01	2284	2398	2501	2512
fun02	23029	24605	25422	26026
fun03	22680	23122	27024	28115
fun04	23112	24342	28521	29139

从总体来看,RetProtect 算法在检测效率、易用性和性能开销方面达到了一个均衡的状态,取得了很好的效果。RetProtect 算法能够很好地针对通过淹没函数返回地址进行的缓冲区溢出攻击进行检测,且效率相比 StackShield 等方法有较大的提升,同时又解决了 StackShield 方法所存在的步骤繁琐、兼容性等方面的问题。

该方法和常见的防止缓冲区地址淹没的几种方法相比有以下的一些优点:

a)该方法与编译器实现自动实现边界检查相比不会大幅降低性能和效率。

b)不会修改函数内存堆栈的布局,减少了出现系统错误和兼容性问题的可能性。

c)在实现了针对函数返回地址的攻击保护之外,避免了 StackShield 方法的一些缺点,效率有了一定的提高,而且兼容性更好。

5　结束语

堆栈溢出仍然是缓冲区溢出漏洞利用中最常见、最普遍的攻击方式。对于针对返回地址的堆栈溢出攻击,本文提出的 RetProtect 算法可以很好地对函数的返回地址进行保护以保证不被恶意修改,相较于其他的针对函数返回地址保护的方法,RetProtect 在保证系统安全性的前提下,采用运行时对返回地址进行备份的方法,同时在最开始的时候结合利用反汇编软件 IDA Pro 对程序进行静态分析,提高了检测效率。本方法在 Linux 平台下对针对返回地址的缓冲区溢出攻击检测有一定的作用,目前能针对单一的攻击方法进行防御,源码修改是针对 GCC4.5.3 进行了改进,对 GCC 的升级版本没有进行测试,但是通过阅读 GCC 的源码发现只要对本文实现的方法稍微进行改进即可实现。下一步的工作主要是将 RetProtect 算法和堆溢出、整型变量溢出等结合起来,以实现同时对多种缓冲区溢出攻击进行检测等。

参考文献:

[1] 国家信息安全漏洞库.信息安全漏洞态势报告(2015 年度)[EB/OL].[2016-01].http://www.cnnvd.org.cn/news/jklbz/id/CNNVD%20Situ ation%20Report.

[2] Cowan C,Wagle F,Pu C,*et al*. Buffer overflows:attacks and defenses for the vulnerability of the decade[J]. DARPA Information Survivability Conference on Exposition,2002,2:119-129.

[3] One A. Smashing the stack for fun and profit[J]. Phrack Magazine,1996,49(7):1996-2006.

[4] 杨少鹏,唐小虎.基于 Linux 的栈溢出攻击防护系统[J].计算机应用研究,2005,22(12):127-130.

[5] Shao Zili,Cao Jiannong,Chan K C C,*et al*. Hardware/software optimization for array & pointer boundary checking against buffer overflow attacks[J]. Journal of Parallel and Distributed Computing,2006,66(9):1129-1136.

[6] Tsai T,Singh N. Libsafe:Transparent system-wide protection against buffer overflow attacks[C]//Proc of International Conference on Dependable Systems and Networks. 2002:541-547.

[7] Cowan C,Pu C,Maier D,*et al*. Stackguard:automatic adaptive detection and prevention of buffer-overflow attacks[C]//Proc of the 7th USENIX Security Symposium. 1998:63-77.

[8] Strackx R,Younan Y,Philippaerts P,*et al*. Breaking the memory secrecy assumption[C]// Proc of the 2nd European Workshop on System Security. New York:ACM Press,2009:1-8.

[9] Marco-Gisbert H,Ripoll I. Preventing brute force attacks against stack canary protection on networking servers[C]//Proc of the 12th IEEE International Symposium on Network Computing and Applications. 2013:243-250.

[10] 刘渊,谢家俊,张春瑞,等.单字节栈溢出的分析[J].计算机工程与设计,2015,36(12):3178-3182.

[11] Chen C M,Chen S M,Ting W C,*et al*. An enhancement of return address stack for security[J]. Computer Standards & Interfaces,2015,38(C):17-24.

[12] Wilander J,Nikiforakis N,Younan Y,*et al*. RIPE:runtime intrusion prevention evaluator[C]//Proc of the 27th Computer Security Applications Conference. New York:ACM Press,2011:41-50.

[13] Pozza D,Sisto R. A lightweight security analyzer inside GCC[C]// Proc of the 3rd International Conference on Availability,Reliability and Security. Washington DC:IEEE Computer Society,2008:851-858.