基于增量链接的PE文件信息隐藏技术研究

田祖伟[1,2] 杨恒伏[1] 罗阳旭[1]
(湖南第一师范学院信息科学与工程系 长沙 410205)[1] (湖南大学信息科学与工程学院 长沙 410082)[2]

摘 要 增量链接旨在提高编译速度和方便程序调试。通过分析采用增量链接后生成的PE文件的特点,提出了一种基于编译器增量链接特性的信息隐藏算法。该方案将隐秘信息隐藏在两个相邻函数代码之间的填充字节中,使得隐藏的信息与程序指令代码紧密结合在一起,极大地提高了隐蔽性和抗攻击性。实验结果表明:该算法隐藏容量大,隐藏信息后的PE文件的长度不会增加,程序性能不受影响,隐蔽性好。

关键词 PE文件,信息隐藏,增量链接,填充字节
中图法分类号 TP309.7 **文献标识码** A

Research of PE File Information Hiding Based on Incremental Link

TIAN Zu-wei[1,2] YANG Heng-fu[1] LUO Yang-xu[1]
(Department of Information Science & Engineering, Hunan First Normal University, Changsha 410205, China)[1]
(School of Information Science and Engineering, Hunan University, Changsha 410082, China)[2]

Abstract Adopting incremental link aims to boost compile speed and make debug convenient. By analyzing PE file characteristics after utilizing incremental link, this algorithm was designed. Concretely, it hides information into padding bytes between adjacent function codes, which integrates hidden information into program instruction codes to guarantee concealment and attack tolerance. The experiment results show that the algorithm has a large hiding capacity, and PE file size does not increase after hidden, and there is not any impact on program performance.

Keywords PE file, Information hiding, Incremental link, Padding byte

1 引言

为了提高链接速度和方便程序调试修改,大多数编译器都提供了增量链接(Incremental Linking)特性[1-5]。链接器不是将所有函数代码都紧挨着放在一起,而是在函数指令代码之间预留一定数量的填充字节(Padding),这样,在程序调试过程中,在函数中增加部分指令就有了存储空间。只要改动不大,没有超过预留填充字节的范围,链接器就不需要全部重新编译,只需要编译所修改的函数即可,这样就可以大大提高链接速度。本文将讨论利用编译器的增量链接特性,将信息隐藏在两个相邻函数指令代码之间的填充字节中。这样,隐藏的信息与指令代码结合在一起,分散隐藏在函数指令代码之间的填充字节中,有效地提高了算法的隐蔽性和抗攻击性。

2 增量链接的工作原理

在编写程序过程中,经常会出现这样一种情况,程序员写了两个函数fun1()和fun2(),这两个函数在源代码中的位置是相邻的,假设函数fun1()的代码从0x400500开始存放,长度为0x200个字节,而函数fun2()的代码紧接着保存在0x400700处。现在需要修改fun1()函数,添加了0x200个字节的指令代码,然后编译了新的代码。函数fun1()的指令代码的长度增加了0x200个字节,现在需要用0x200字节来保存新增加的代码,链接器怎么进行处理呢?常规的方法是重新编译链接,将已编译好的exe删除,然后重新生成所有的函数代码,即fun2()函数向后移动0x200字节的位置,给fun1()腾出空间来,fun1()之后所有的函数全部重新定位。对于大型软件来说,这个重新编译链接的时间开销是比较大的,程序员不可避免地需要不断地调试修改代码,不断地重复这个耗时的工作。

为了提高链接速度和方便程序的调试修改,许多编译器都引入了增量链接特性。其主要工作原理是:链接器不是将所有函数代码都紧挨着放在一起,而是在相邻的两个函数之间加上填充区域,这样函数添加部分指令就有了存储空间。只要改动不大,没有超过填充区域的范围,链接器就不需要编译链接,这样大大提高了链接的速度。如果对函数的修改较大,增加的代码超过了填充区域的范围,链接器将在节的末尾增加一个较大的填充区域,将修改的函数移动到节的末尾,同时修正所有调用此函数的CALL指令,进行重定位。

为了解决这个问题,在代码节中引入了一个ILT表(Incremental Linking Table)[5-10],其工作原理如下。

将之前直接调用函数的指令如:

CALL FUN1

到稿日期:2012-02-11 返修日期:2012-07-01 本文受国家自然科学基金项目(61073191),湖南省自然科学基金项目(11JJ3075),湖南省科技计划项目(2011GK3139),湖南省普通高等学校教学改革研究项目(2012[528]),湖南省大学生研究性学习和创新性实验计划项目(2010[421])资助。

田祖伟 男,博士,教授,主要研究方向为信息安全、算法分析与设计、编译优化;杨恒伏 男,博士,副教授,主要研究方向为信息安全、图像处理。

改为:CALL FUN1_STUB

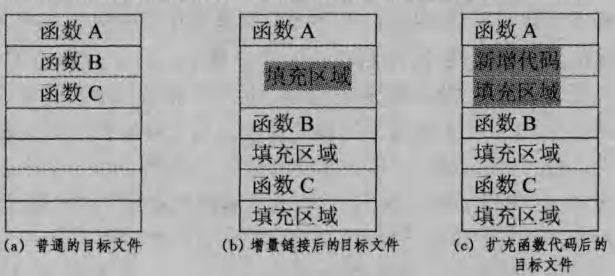
FUN1_STUB:JMP FUN1

这样,如果函数 FUN1()的改动很大,链接器直接将函数 FUN1()移动到节的末尾,然后只需要修改这个 JMP 指令即可,可以看到,这种实现方式开销是 O(1)。当很多个函数都用这种方式时,就形成了一个有 JMP 指令构成的表——ITL 表。图 1 为 Helloworld. exe 文件中的 ILT 表。

```
012C1000 CC          int 3
012C1001 CC          int 3
012C1002 CC          int 3
012C1003 CC          int 3
012C1004 CC          int 3
@ILT+0(__setdefaultprecision):
012C1005 E9 56 1F 00 00 jmp   _setdefaultprecision (12C2F60h)
@ILT+5(__RTC_GetErrDesc):
012C100A E9 81 1C 00 00 jmp   _RTC_GetErrDesc (12C2C90h)
@ILT+10(_RegisterClassExW@4):
012C100F E9 10 0B 00 00 jmp   RegisterClassExW (12C1B24h)
@ILT+15(@__security_check_cookie@4):
012C1014 E9 C7 0D 00 00 jmp   __security_check_cookie (12C1DE0h)
@ILT+20(_IsDebuggerPresent@0):
012C1019 E9 02 2C 00 00 jmp   IsDebuggerPresent (12C3C20h)
@ILT+25(_GetStartupInfoW@4):
012C101E E9 F1 2B 00 00 jmp   GetStartupInfoW (12C3C14h)
@ILT+30(_EndDialog@8):
012C1023 E9 44 0B 00 00 jmp   EndDialog (12C1B6Ch)
@ILT+35(__RTC_Terminate):
012C1028 E9 13 20 00 00 jmp   _RTC_Terminate (12C3040h)
@ILT+40(_LoadAcceleratorsW@8):
012C102D E9 E6 0A 00 00 jmp   LoadAcceleratorsW (12C1B18h)
......
```

图 1　Helloworld. exe 文件中的 ILT 表

ILT 表的引入是为了加快编译速度,编译器在代码节的首部创建一个 ILT 表来间接调用函数,ILT 表中的每一个元素都是一条长度为 5 字节的 JMP 指令(第 1 个字节为 JMP 指令的汇编代码,后面 4 个字节记录函数的真实地址),通过 ILT 跳转到函数的实际地址,这样,在某个函数地址发生变化时,编译器只需要修改 ILT 表,而不用修改每个调用函数的语句,这样,可以大大提高编译速度,方便调试。在增量编译模式下,为方便添加新的函数,在 ILT 表和可执行程序的第一个函数目标代码之间填充大量字节(0xCC),这样,新添加的函数会顺序往后填写这个表格,以前填写过的保持不动,这样达到加快编译的速度。其增量链接工作原理如图 2 所示。



(a) 普通的目标文件　(b) 增量链接后的目标文件　(c) 扩充函数代码后的目标文件

图 2　增量链接工作原理

在 VS2010 编译一次代码,然后用 IDA 或者 W32Dasm 之类的软件可以看到两个函数之间间隔了不少距离,而这些间隔即填充区域。填充区域全部以数据 0xCC(指令 INT 3 的机器码)进行填充。在 Windows 下,执行 INT 3 指令将会引发一个异常,程序会被终止或是返回调试器。这样,如果在前一个函数中增加部分程序代码,编译后可以看到两个函数在目标文件中的位置不变,但函数间的填充区域变小了,增加的部分程序代码将覆盖填充区域,如图 2 所示。

3　基于增量链接的数据隐藏算法

Visual C++预定义提供了两组编译选项的集合,即 Debug 版本和 Release 版本,编译器按照预定的选项进行编译和链接。通常称 Debug 为调试版本,通过一系列编译选项的配合,编译的结果通常包含调试信息,而且不做任何优化,为程序员提供了强大的应用程序调试能力。Release 也称为发行版本,一般不允许在发行版本上进行调试,所以不包含调试信息,它往往是进行了各种编译优化,使得程序在代码大小和运行速度上都是最优的,以便用户很好地使用。Visual C++的链接器有许多选项,其中/INCREMENTAL:XX(XX 可以是 YES 或 NO)选项控制链接器如何处理增量链接(Incremental Linking)。增量链接是 Debug 模式的缺省设置,当采用此模式时,链接器只重写上次链接之后改动过的部分代码,允许链接器增加函数和数据的大小而不用重新创建. exe 文件,这样可以减少链接程序的工作,显著提高链接器的性能,使得 Visual C++可以在快速完成链接的同时,方便程序员调试程序。

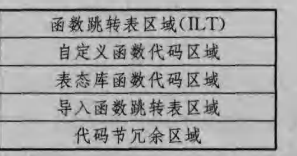


图 3　PE 文件代码节内部数据组织(Debug 版本)

为了实现这个目的,在增量链接模式下,Visual C++链接器填充了大量的 INT 3 指令(机器码为 0xCC,在 Intel 系统中,0xCC 的含义是 int 3 中断——如果不小心执行了这块数据,那么程序会马上中断并且提示用户,其他的值则是典型的非法数据)到可执行文件的函数与函数之间。因此当对源代码进行修改,增加函数或语句后,增加的代码就覆盖在函数与函数之间填充的 INT 3 所在的空间中。可执行文件的其他部分不需要进行任何改动,但是增量链接是有代价的,即会增加可执行文件的长度。一个使用增量链接的文件大约有 1/3 的空间来存放 INT 3。在大的执行文件中,甚至可能增加几百 KB 甚至几 MB 的 INT 3 填充字节。在这种文件里,每个文件里的函数还多了一个 JMP 指令。当调用增量链接文件里的函数时,首先通过 CALL 指令找到相应的 JMP,然后通过 JMP 指令执行所需要的函数。这些 JMP 的好处是使链接程序可以在内存中任意移动函数而不用更新使用这些函数的 CALL 指令,这些空间可用于信息隐藏。

3.1　基于增量链接的信息嵌入算法

在 Visual C++中,采用 Debug 模式进行编译链接,系统默认打开增量链接开关(即/INCREMENTAL:YES),这样将在生成的目标文件的函数与函数之间填充大量的字节(0xCC),这些填充字节可以用于隐藏信息。图 4 为信息嵌入过程示意图。

详细的信息隐藏算法描述如下。

步骤 1　分析 PE 文件代码节,计算填充字节的总容量。

步骤 2　利用密钥 *key* 对原始秘密信息 W 进行加密,得到最终待隐藏秘密数据 W'。

步骤 3　分析 ILT 表,计算跳转表中目标的地址(函数在磁盘文件中的首地址),并按目标地址从大到小进行排序(目标地址最大的即为代码节中最后一个函数)。在 ILT 表中,

每个 JMP 指令占 5 个字节，为一条远转移指令，采用相对寻址，后面 4 个字节为相对地址，设 ILT 表中某个 JMP 指令在磁盘文件中的偏移地址为 X，JMP 指令中给出的相对地址为 d，则相应目标函数的入口地址的计算如下：

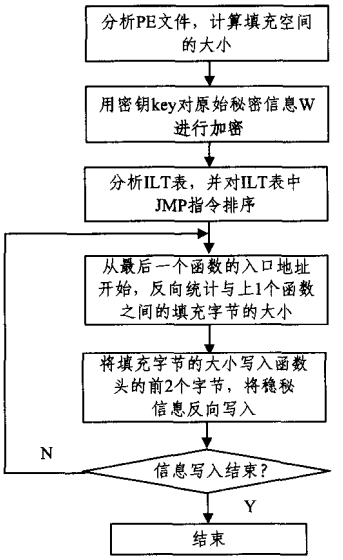


图 4　信息嵌入过程示意图

函数入口地址=X+d+5，其中 5 为 JMP 指令本身的长度。

步骤 4　从最后一个函数的入口地址的前 1 个字节开始，先反向统计填充字节 0xCC 的个数，并将此数据写入此填充块的最后两个字节中（即函数入口地址的前 2 个字节中），然后将待隐藏秘密数据 W' 按顺序，根据填充字节的大小按从高地址到低地址的方向反向写入相应单元中。

步骤 5　当一个填充块使用完后，根据已排序的函数入口地址，跳转到下一个函数，重复第 4 步，一直到第一个函数。这样可以得到含有秘密信息的 PE 文件。

3.2　基于增量链接的数据提取算法

对于给定的含有隐秘信息的 Debug 版的 PE 文件，不需要给出原始的 PE 文件，就可以很容易地提取隐藏信息。

数据提取过程由以下步骤组成：

步骤 1　对于给定的 Debug 版的 PE 文件，首先分析 ILT 表，计算跳转表中目标地址（函数在磁盘中的首地址），并按目标地址从大到小进行排序（目标地址最大的即为代码节中最后一个函数）。

步骤 2　读取最后一个函数的入口地址的前 2 个字节的值 N，从距离函数首地址偏移为 3 的单元开始，从高地址到低地址的方向反向提取 N 个地址单元的值。

步骤 3　一个填充块提取完成以后，根据已排序的函数入口地址，跳转到下一个函数，重复第 2 步，直到第一个函数。这样可以得到隐藏在 PE 文件中的加密信息。

步骤 4　利用密钥 key 对加密信息 W' 进行解密得到原始的信息 W。

4　实验结果

实验中，采用 Debug 版的 Helloworld. exe 作为载体文件，该程序功能简单，仅在窗口中间显示“Hello，World”字样，使用 Vistual studio 2010 的 Win32 debug 模式进行编译，生成的可执行的 PE 文件 helloworld. exe 大小是 88,576 bytes。对生成的 helloworld. exe 文件，利用二进制编辑器，可以观察到该 PE 文件内部代码节的原始数据组成形式。如图 5 所示，文件地址从 400H 到 63FH 之间为函数跳转表，即 ILT 表；文件地址从 640H 到 86FH 为 ILT 表与第一个函数_tWinMain() 函数之间的填充字节（用 0xCC 填充）。文件地址从 870H 到 9B4H 之间为_tWinMain() 函数的指令代码；文件地址从 9C0H 到 9FFH 之间为第一个函数_tWinMain()与第二个函数 MyRegisterClass()之间的填充字节。文件地址从 A00H 到 AF8H 之间为 MyRegisterClass()函数的指令代码。

实验表明，在 helloworld. exe 程序中，填充字节多达 2190 个字节，可隐藏信息达到 1960 多个字节。

```
Offset(H) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
; helloworld.exe 文件地址 400H-63fH 之间为函数跳转表，即 ILT 表。
00000400  CC CC CC CC CC E9 56 1F 00 00 E9 81 1C 00 00 E9  ìììlìéV...é....é
00000410  10 0B 00 00 E9 C7 0D 00 00 E9 02 2C 00 00 E9 F1  ....éÇ...é.,...éñ
00000420  2B 00 00 E9 44 0B 00 00 E9 13 20 00 00 E9 E6 0A  +..éD...é. ..éæ.
00000430  00 00 E9 E3 2B 00 00 E9 F4 0B 00 00 E9 0F 1D 00  ..éã+..éô...é...
00000440  00 E9 3A 1C 00 00 E9 15 0B 00 00 E9 92 1F 00 00  .é:...é....é' ..
00000450  E9 AD 2B 00 00 E9 38 2C 00 00 E9 81 21 00 00 E9  é.+..é8,..é.!..é
00000460  F2 2B 00 00 E9 9B 22 00 00 E9 F8 0A 00 00 E9 23  ò+..é?"..é?...é
00000470  25 00 00 E9 20 2C 00 00 E9 23 21 00 00 E9 A4 2B  %..é ,..é#!..é¤+
00000480  00 00 E9 8B 0A 00 00 E9 24 1C 00 00 E9 9F 09 00  ..é?...é$...é?..
 ……
00000630  E9 FB 08 00 00 E9 74 03 00 00 E9 69 03 00 00 CC  é?...ét...éi...ì
; helloworld.exe 文件地址从 640H 到 86FH 为 ILT 表与第一个函数
; _tWinMain() 函数之间的填充字节。
00000640  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000650  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000660  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000670  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000680  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000690  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
……
00000850  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
00000860  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC  ìììììììììììììììì
; 文件地址从 870H 到 9B4H 之间为_tWinMain() 函数的指令代码
00000870  55 8B EC 81 EC F0 00 00 00 53 56 57 8D BD 10 FF  U?ì.ìð...SVW.?.?
00000880  FF FF B9 3C 00 00 00 B8 CC CC CC CC F3 AB 8B F4  ??¹<...?ìììó???
00000890  6A 64 68 00 72 41 00 6A 67 8B 45 08 50 FF 15 C0  jdh.rA.jg?E.P?.À
……
```

图 5　Helloworld. exe 代码节函数间的填充字节

（下转第 132 页）

次执行中验证多条 LTL/CTL 公式。本文研究为开发更为高效的并发系统验证工具提供了新思路。下一步除了不断完善本文提出的验证框架之外，还将进一步通过更多的实验对该方法的效率进行测试和对比，研究在待验证的性质不满足时给出反例，及基于所给反例进行系统诊断的技术。

参 考 文 献

[1] Hoare C A R. Communicating Sequential Processes [M]. http://www.usingcsp.com/cspbooks,2004

[2] Garavel H, Lang F. NTIF: A General Symbolic Model for Communicating Sequential Processes with Data [C]// Proc. FORTE'02. 2002:276-291

[3] Clarke E M, Talupur M, Veith H. Proving Ptolemy Right, The Environment Abstraction Framework for Model Checking Concurrent Systems [C]// Proc. TACAS. 2008:33-47

[4] Chaki S, Clarke E M, Ouaknine J, et al. Concurrent software verification with states, events, and deadlocks[J]. Formal Aspects of Computing, 2005, 17(4):461-483

[5] Dubrovin J. Efficient Symbolic Model Checking of Concurrent Systems [D]. Aalto University School of Science Department of Information and Computer Science, Doctoral Dissertation, 2011

[6] 张兆庆，蒋昌俊，等. PVM 并行程序验证系统的原理与实现[J]. 计算机学报，1999，22(4)：409-414

[7] Gelfond M, Lifschitz V. The stable model semantics for logic programming[C]// Proc. ICLP'1988. 1988:1070-1080

[8] Baral C. Knowledge Representation, Reasoning, and Declarative Problem Solving [M]. Cambridge Press, 2003:5-64

[9] Zhao Ling-zhong, Qian Jun-yan, Chang Liang, et al. Using ASP for Knowledge Management with User Authorization [J]. Data & Knowledge Engineering, 2010, 69(8):737-762

[10] Gavanelli M, Rossi F. Constraint logic programming [C]// LNCS. 2010, 6125:64-86

[11] De Angelis E, Pettorossi A, Proietti M. Synthesizing Concurrent Programs using Answer Set Programming [C]// Proc. CS&P 2011. 2011:85-98

[12] Heljanko K, Niemelä I. Answer set programming and bounded model checking [C]// Proc. AAAI Spring 2001 Symposium on Answer Set Programming. 2001:90-96

(上接第 93 页)

5 性能分析与讨论

5.1 隐蔽性分析

采用基于增量链接的隐藏算法，将隐秘信息嵌入到函数指令代码之间的填充字节中、使得隐藏的信息与原程序中函数的指令代码融合在一起、其隐蔽性高。同时，在此方法中，不需要修改 PE 文件结构中的其它数据，不会增加文件长度，也不会影响程序性能，大大提高了隐蔽性。

5.2 嵌入容量分析

对 Windows 窗口应用程序，编译器将自动添加众多的初始化函数和窗口框架函数，函数指令代码之间的填充空间多，隐藏容量大。隐藏容量与应用程序功能有关，一般来说，程序功能越复杂，程序中的函数(包括用户自定函数和静态链接库函数)越多，填充空间越大，隐藏容量也就越大。表 1 为基于增量链接的信息隐藏方案隐藏容量分析。

表 1 基于增量链接的信息隐藏方案隐藏容量分析

文件名	文件大小 (bytes)	ILT 表的大小(bytes)	文件中函数个数	填充字节 (bytes)	嵌入容量 (bytes)
helloworld. exe	88576	575	115	2190	1960
hideinfo. exe	245248	4540	908	19170	17354
inforemix. exe	254026	315	63	2651	2525
movecode. exe	286797	895	179	11866	11508
iocnHide. exe	135263	310	62	6467	6343
import. exe	110686	185	37	1768	1694

5.3 鲁棒性

PE 文件结构复杂，代码节数据组织更为复杂，并且 PE 文件与图像、文本文件不一样，即使是一个字节的修改都可能导致程序不能运行。此方法隐藏信息后，文件长度不会增加，程序运行正常，性能也不受影响。

结束语 现有 PE 文件隐藏算法存在隐藏信息过于集中、没有与程序指令代码结合在一起、安全性不高等问题。本文利用编译器增量链接的特性，将隐秘信息隐藏在函数代码之间的填充字节中，使得隐藏的信息与程序指令代码紧密结合在一起，极大地提高了抗攻击性。由于对 Windows 窗口应用程序，编译器将自动添加众多的初始化函数和窗口框架函数，因此函数指令代码之间的填充空间多，隐藏容量大。隐藏信息后的 PE 文件的长度不会增加，程序性能不受影响，隐蔽性好。

参 考 文 献

[1] Appel A W, Ginsburg M. 现代编译原理：C 语言描述[M]. 赵克佳，黄春，沈志宇，译. 北京：人民邮电出版社，2006

[2] Aho A V, Lam M S, Sethi R, et al. Compilers: Principles, Techniques, and Tools (2nd Edition)[M]. Addison Wesley, 2008

[3] 俞甲子，石凡，潘爱民. 程序员的自我修养—链接、装载与库[M]. 北京：电子工业出版社，2009

[4] Levine J R. Linkers and Loaders[M]. Morgan Kaufmann, 2005

[5] 关于 Visual C++增量链接以及. textbss[OL]. http://www.cnblogs.com/Dahaka/archive/2011/08/01/2124256.html, accessed September, 2012

[6] Pietrek M. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format[J]. Microsoft Systems Journal, 1994, 9(3):15-34

[7] Richter J, Nasarre C. Windows 核心编程[M]. 葛子昂，周靖，廖敏，译. 北京：清华大学出版社，2008:509-522

[8] Pietrek M. Windows 95 System Programming Secrets[M]. 候俊杰，译. 台湾：旗标出版社，1997:559-623

[9] 陈凯明，刘宗田. 反编译研究现状及进展[J]. 计算机科学，2001，28(5)：113-115

[10] Cifuentes C. Reverse Compilation Techniques [D]. School of Computing Science, Queensland University of Technology, July 1994