文章编号:1001 - 9081(2005)03 - 0623 - 02

反静态反汇编技术研究

吴金波,蒋烈辉
(信息工程大学 信息工程学院,河南 郑州 450002)
(wjinbo@163.com)

摘 要:通过对软件可执行二进制码的静态反汇编结果进行分析,可以对其进行非法的修改或窃取其知识产权。为了防范这种情况,在描述静态反汇编基本算法的基础上,提出了分支函数和跳转表欺骗两种隐藏程序控制流的反静态反汇编技术。这两种技术能够隐藏程序中跳转指令的真实目标地址,并能够伪造出导致静态反汇编器出错的假目标地址,从而提高程序的反静态反汇编性能,增加软件分析的难度。

关键词:可执行程序;控制流隐藏;反静态反汇编
中图分类号: TP314 **文献标识码**:A

Research on resistance to static disassembly

WU Jin-bo, JIANG Lie-hui
(*College of Information Engineering, Information Engineering University, Zhengzhou Henan* 450002, *China*)

Abstract: The goal of making unauthorized modifications or stealing intellectual property of software can be reached by analyzing the static disassembly result of its executable binary code. In order to avoid this kind of instance, this paper described the basic static disassembly algorithm, then proposed two anti-disassembly techniques focusing on the obfuscation of control flow, branch function and jump table spoofing. These two techniques can hide the real target addresses of jump instructions, and can fabricate target addresses which will lead to error of static disassemblers. Therefore the program's resistance to static disassembly will be improved, and it is more difficult to analyze the software.

Key words: executable code; obfuscation of control flow; resistance to static disassembly

0 引言

目前,大多数的软件都是以可执行代码的形式发布的,利用静态反汇编工具可以容易地对其进行分析,发现其漏洞及核心算法,进而进行非法的修改或窃取其知识产权。为了防范这种非法行为,许多软件采用了诸如程序指纹、程序水印等技术,但这些技术还是不能非常有效地保护软件不被攻击和盗版。静态反汇编工具同样能对实现这些程序保护技术的程序进行分析,从而隔离或破坏这些水印和指纹,使其防范作用失效。但是,如果程序本身具有反静态反汇编的性能,就可以增加攻击者分析可执行代码的难度,从而起到软件保护的作用。

通过对静态反汇编器实现的基本算法以及大量程序的汇编级代码进行研究以后发现:利用一种我们称之为控制流隐藏的技术可以达到阻止或干扰静态反汇编器进行有效反汇编的目的,即可以使可执行程序本身具有反静态反汇编的性能。"控制流隐藏"技术是通过对汇编代码级的程序使用一定的方法进行改造,尽可能地隐藏程序中确定程序静态控制流程的关键点或伪造一些与程序控制流有关的关键点来迷惑静态反汇编工具,使之得到不正确的静态程序控制流程,从而得到不正确的反汇编结果。使用这种技术对程序进行改造以后,不会改变程序原有的动态执行结果,并且不对程序的效率产生大的影响。这种控制流隐藏技术就是本文所讨论的反静态反汇编技术。

1 静态反汇编的基本策略

静态反汇编是指在对可执行代码进行反汇编的过程中,不执行相关的代码,通过对代码的静态分析得到汇编程序,从而获得程序功能的方法。目前应用最广的静态反汇编策略是基于程序静态控制流程的递归策略,下面就对这种策略进行简要的介绍。

1.1 基于程序静态控制流的递归扫描策略

递归的反汇编扫描策略中是以程序静态控制流程为基础,通过对可执行代码的扫描来获得比较准确的反汇编结果。在这种策略中,每一条改变程序流程的指令(例如:跳转指令、调用指令等)是反汇编器扫描的关键点,当扫描到这些指令时,以它们跳转或调用的目标地址作为某一个新程序段的起始;而程序返回类指令等表示程序结束的指令是反汇编器扫描的另一个关键点,当扫描到这类指令时表示当前正在扫描的是程序段的结束。通过使用这样的策略,可以有效地跳过嵌入在程序段中的数据以及无效代码。

该策略的大体流程为:

```
global startaddr, endaddr
void disasmrec ( addr)
{
  while( startaddr < = addr < = endaddr)
  {
    If ( addr has been visited already)
      return;
```

收稿日期:2004 - 08 - 29;修订日期:2004 - 12 - 16

作者简介:吴金波(1979 -),男,江苏武进人,硕士研究生,主要研究方向:计算机应用; 蒋烈辉(1967 -),男,浙江东阳人,教授,硕士,主要研究方向:计算机应用、嵌入式系统.

```
    I = 在地址 addr 处翻译所得的指令;
                                  // I 是指向描述指令结构体的指针
    标记 addr 为已访问的地址;
    If( I 是一个跳转或调用指令)
    {
      For ( I 的每一个可能的目标 t)  do
        disasmrec ( t);
      return;
    }
    else addr + = length( I);
  }
}
```

该策略的实现是基于这样一个假设:即能够十分准确地获得所反汇编的目标文件的静态控制流程。然而,若对反汇编目标程序进行适当处理,可以使静态控制流不那么明显,这将导致反汇编器的反汇编过程无法正常有效地进行,所获得的反汇编结果必然也是不准确的。

1.2 跳转表恢复技术

为了处理目标二进制码中的间接跳转的问题,一种叫作跳转表恢复的技术应用到了静态反汇编器中。跳转表是由一系列目标地址所组成的特殊数据段(其形式见图 1),该表中的表项内容将作为某一条或几条间接跳转指令的目标操作数。它在程序中出现的一般形式如程序清单 1 所示。

程序清单 1

```
r = evaluate i;
if r > = N goto default;
r * = 4;
r + = BaseAddr;
jmp * r;
```

跳转表恢复技术通过使用二进制可执行代码分片(binary slicing)以及表达式替换技术(expression substitution)来获得可执行代码中跳转表的基地址 BaseAddr 以及跳转表的项数 N,由于在正常情况下跳转表中的所有项都是某一个程序的起始地址,故以这些地址为起始就可以获得某些程序段的准确反汇编结果。

这种技术也有一定的缺点:它不能识别跳转表中的无效项(即在任何情况下也不可能作为跳转指令目标操作数的地址),若这些无效项指向数据段或某条指令的中间位置,将导致反汇编器产生十分严重的异常。

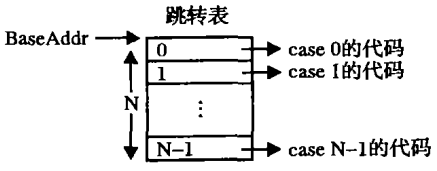

图 1 跳转表的格式

2 控制流隐藏技术

控制流隐藏技术主要是利用以上静态反汇编技术的弱点,在不改变程序动态执行顺序的前提下,通过对程序进行一定的修改来隐藏程序的静态控制流程,以此来迷惑静态反汇编工具,使之获得不正确的反汇编结果,从而提高程序的反静态反汇编性能。

如上所述,基于程序静态控制流的递归扫描反汇编器的最大特点是:它可以抽取程序的静态控制流,并以此为依据跳过镶嵌在程序段中的数据,产生较为准确的反汇编结果。但是反汇编器获得程序控制流的方法不是非常可靠,具体来说有以下两点:第一点,递归扫描反汇编器在遇到控制流转移指令时,将处理所有可能的目标地址,无法判断跳转指令本身及其目标地址的合法性;第二点,静态反汇编器处理间接转移或间接调用指令是十分困难的,只能通过跳转表恢复或随机反汇编的方法来处理,其准确性更无法得到保证。所以,若进行控制流隐藏,则可以使静态反汇编器无法有效地进行反汇编。下面介绍几种行之有效的控制流隐藏技术。

2.1 分支函数

无条件跳转指令是反汇编器确定程序控制流程的主要依据之一,遇到这类指令表示当前程序段结束,并且转移到跳转指令的目标地址继续反汇编。而程序调用指令不能表示程序的结束,其后所跟的仍然是当前处理的程序段。根据这个特性,若将无条件跳转指令改造成为对某一个能完成相同功能的函数的调用指令,将会影响反汇编器对紧随无条件跳转指令后的二进制码的反汇编;如果能让所调用的函数隐藏本身的流程,静态控制流程隐藏的效果将更明显。我们将能够完成无条件跳转指令相同功能的函数叫作分支函数。

给出分支函数的定义。首先定义一个有限的关于程序地址的映射,表示的是无条件跳转指令地址与其跳转目标地址之间的对应关系:

$\varphi = \{addra1 \rightarrow addrb1, \cdots, addran \rightarrow addrbn\}$

所谓的分支函数 $f\varphi$ 是这样一个函数:在某个地址 addra$i$ 调用它以后将导致程序的控制流转移到与其对应的地址 addrb$i$,使用分支函数可以将程序中 n 个地址处调用的无条件跳转指令(如程序清单 2 所示)转化为统一的对分支函数的调用(如程序清单 3 所示)。由此可以得出:分支函数所完成的工作是根据当前调用指令的地址 addra$i$ 将程序的执行流程转移到对应的目标地址 addrb$i$,故分支函数不是返回到调用指令的下一条指令,而是返回到 addrb$i$。

程序清单 2

```
addra1: jmp addrb1
...
addra2: jmp addrb2
...
addran : jmp addrbn
```

程序清单 3

```
addra1: call fφ
...
addra2: call fφ
...
addran : call fφ
```

分支函数有多种实现方法,为了不对原有程序的执行效率产生过大的影响并隐藏分支函数的程序行为,我们使用了基于 Hash 函数的实现方法,原因是 Hash 函数本身可以隐藏程序流,同时其时间复杂度也只有 $O(1)$。

以下是具体的实现:

1) 确定程序中需要进行变换的地址集合 {addra1, addra2, …, addran},以及与其对应的跳转目标地址集合 {addrb1, addrb2, …, addrbn}。

2) 构造一个从集合 {addra1, addra2, …, addran} 到集合 {1, 2, …, n} 的 Hash 函数 h,并产生一个数组 T[n]用于存储调用地址与目标地址之间的差值。

3) 对于每一个 {addra$i$, addrb$i$} 序偶,按以下的公式将 addrb$i$ - addra$i$ 的值存入到数组 T[n]中:T[h(addra$i$)] = addrb$i$ - addra$i$。

4）产生一个分支函数 f,其入口参数为跳转指令的地址 addrai,根据此地址调用 Hash 函数产生在 T 中的索引值,获得 T[h(addrai)]的值,加上 addrai 得到跳转的目标地址 addrbi。通过具有隐藏程序静态控制流特性的方法(如修改堆栈的返回地址)在函数 f 中将程序的流程转移到 addrbi。

5）将以上产生的 Hash 函数 h、分支函数 f 以及数组 T 添加到原来的程序中,并将每个 addrai 地址处的无条件跳转指令修改为对函数 f 的调用指令,从而完成对分支函数的添加。

分支函数有以下优点:

首先,分支函数可以非常有效地隐藏程序的控制流程。通过使用以上所述的基于 Hash 函数的分支函数,反汇编器将无法获得这些修改以后的无条件跳转处所产生的程序控制流的变换。

其次,分支函数可以更显著地迷惑反汇编器。由于反汇编器在遇到子程序调用指令时,将在该指令之后继续进行反汇编,可以在所有调用分支函数的指令之后加入能导致反汇编器产生异常的无效代码(例如在这些指令之后插入伪跳转指令,将程序的静态流程伪装到某个数据段中)。同时,由于分支函数完成的是无条件跳转指令的功能,所以这些代码的添加将不会影响程序的动态行为。

分支函数也有一定的缺点:由于对分支函数的调用涉及到现场的保存与恢复,将会对程序的执行效率产生影响。所以,这种方法用于动态执行次数不太多的程序段比较好。

2.2 跳转表欺骗

静态反汇编器一般都使用跳转表恢复技术来解决由 switch…case…语句产生的间接跳转。在这种技术中,反汇编器根据间接跳转指令的上下文,获得跳转表的起始位置和边界,将其中的表项作为程序的起始地址并从此处开始一次反汇编过程,获得对应程序段的汇编代码。我们可以利用跳转表恢复技术的这种特性,使用由不透明表达式伪造的跳转表来迷惑静态反汇编器。

首先介绍不透明表达式(opaque expressions),所谓不透明表达式就是指含有寄存器或内存单元的内容作为输入且输出结果恒为某个常数的表达式,这样的表达式结果值对于静态反汇编器来说是不可知的。若在程序中使用输出结果恒定为 k 的不透明表达式作为基于跳转表的间接跳转的条件判断指令,则间接跳转的目标地址在跳转表中的相对偏移量 r 是恒定的,所以在跳转表中只有一项是实际有效的跳转目标地址,其他都是无效项。由此,可以在跳转表的其他 $N-1$(N 为跳转表的项数)项都填入数据段地址或某条指令中间地址等非法地址。反汇编器以这些地址为程序段的起始地址启动一次反汇编过程必然会导致异常或获得不准确的反汇编结果。

跳转表欺骗技术的具体实现方法如下:

1）在程序段中寻找合适的无条件跳转指令,取出其跳转的目标地址 TargetAddr,作为跳转表中唯一有效项的内容;

2）确定跳转表的项数 N 以及跳转表的基地址 BaseAddr;

3）构造输出恒为 k 的不透明表达式,并根据 k 的值确定跳转表中有效项的序号 m,在跳转表的第 m 项中填入 TargetAddr;

4）在跳转表的其他 N-1 项中填入数据段地址或某条指令中间的地址,完成对伪跳转表的构造,将跳转表插入到原程序中;

5）按照程序清单 1 所示构造一个基于跳转表的间接跳转。

跳转表欺骗的方法有以下优点:

首先,对于不能处理间接跳转的反汇编器来说,这种方法非常有效地隐藏程序的静态流程;即使是能处理跳转表的静态反汇编器也能通过在跳转表中伪造非法的表项,误导反汇编器,使其产生异常。

其次,原程序在改造完以后,程序执行的效率基本不发生变化,所以这种方法可以用于改造一些执行频率较高的程序段中的无条件跳转指令,从而更有效地隐藏程序的流程。

参考文献:

[1] AUCSMITH D. Tamper-resistant software: An implementation[A]. Information Hiding: First International Workshop: Proceedings, LNCS1174[C]. Springer-Verlag, 1996.317-333.

[2] CHO W, LEE l, PARK S. Againt intelligent tampering: Software tamper resistance by extended control flow obfuscation[A]. Proceedings of World Multiconference on Systems, Cybernetics, and Informatics[C], 2001.

[3] CIFUENTES C, Van EMMERIK M. Recovery of jump table case statements from binary code[J]. Science of Computer Programming, 2001,40(2/3):171-188.

[4] COLLBERG C, THOMBORSON C, LOW D. Manufacturing cheap, resilient, and stealthy opaque[A]. Proceedings of 25th ACM Symposium on Principles of Programming Languages (POPL 1998)[C]. 1998.184-196.

[5] OGISO T, SAKABE Y, SOSHI M, *et al*. Software obfuscation on a theoretical basis and its implementation[J]. IEEE Transactions on Fundamentals, 2003, E86-A(1).

[6] SCHWARZ B, DEBRAY S, ANDREWS G. Disassembly of executable code revisited[A]. Working Conference on Reverse Engineering[C], 2002.

(上接第 619 页)

向哪个文件,只要使用指针最高位的若干个比特就能实现。

2.4 更新算法

数据更新一般是利用 FLPM-B * 树的插入或者删除算法来实现的。如果更新的数据量较少,也可以采用更加快捷的办法。删除节点时把节点的 Num 标识设置为 0 并链入自由节点串。插入新数据时,如果不希望进行 FLPM-B * 树数据节点的重新排序,或者不希望申请新的节点,那么可以使用一个临时文件,这时不生成索引节点,只生成数据节点。在检索时对该临时文件进行一次扫描。在系统空闲时再把临时文件中的数据插入到 FLPM-B * 树中。这只是一种应急措施,以便快速进行数据更新并提供检索。

3 结语

FLPM-B * 树的算法设计充分利用了 FLPM-B * 树模块化结构的特点,实现了 FLPM-B * 树重构、整理与分割等特殊功能。同时说明 FLPM-B * 树具有一定的故障恢复能力,能够方便地对保存该树的磁盘文件进行分割,可以进行整理以充分利用存储空间并保证磁盘操作的连续性。FLPM-B * 树的这些特点可以显著改善系统的性能。

参考文献:

[1] 袁向阳. 基于二元语法的科技档案信息检索分类技术研究[D]. 国防科学技术大学, 2004.

[2] 刘祖斌, 王永成, 刘椿年. 中文全文检索系统中压缩模型和模式匹配技术[J]. 中文信息学报, 2000,14(4):42-47.