计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140607
52(8):1902-1909,2015

基于代码移动的二进制程序控制流混淆方法

陈 喆 王 志 王晓初 贾春福
(南开大学计算机与控制工程学院 天津 300071)
(chenzhe501@foxmail.com)

Using Code Mobility to Obfuscate Control Flow in Binary Codes

Chen Zhe, Wang Zhi, Wang Xiaochu, and Jia Chunfu
(*College of Computer & Control Engineering, Nankai University, Tianjin* 300071)

Abstract Code obfuscation is usually used in software protection and malware combating reverse engineering. There are some security issues in traditional code obfuscation methods, because reverse engineers can acquire all binary codes. To mitigate this problem, this paper presents a novel control flow obfuscation approach to protect the control flow of binary codes based on code mobility. Transforming the significant control logic codes to a remote trusted entity beyond adversary's control makes some control flow information invisible at local untrusted execution environment, so that the binary code's key behaviors cannot be predicted statically or dynamically. Non-conditional jump instructions without control information are used to replace some critical conditional jumps to hide branch conditions and jump target memory addresses, which increases the difficulty of collecting and reasoning about the program path information. We estimate this obfuscation approach in three aspects: potency, resilience and cost. And using this approach, we obfuscate the trigger conditions in six malware samples belonging to different families, and then use the state-of-the-art reverse engineering tools to reason about their internal control logic. Experimental result shows that our obfuscation approach is able to protect various branch conditions and reduce the leakage of branch information at run-time that impedes reverse engineering based on symbolic execution to analyze program's internal logic.

Key words code obfuscation; control-flow obfuscation; code mobility; reverse engineering; symbolic execution

摘 要 代码混淆技术常被用于软件保护领域和恶意代码对抗分析.传统的代码混淆技术会使逆向分析者获得程序的全部二进制代码,因此存在一定的安全性问题.为缓解这一问题,提出了一种基于代码移动的二进制程序控制流混淆方法,将程序的重要控制逻辑代码移动至逆向分析者不可控的可信实体,以使本地代码控制流信息部分缺失,从而使得程序的关键行为无法通过推理获知;利用包含无初始意义操作数的非条件跳转指令替代条件跳转指令隐藏路径分支的分支条件和目标地址,以增大收集程序路径信息的难度.对该控制流混淆方法从强度、弹性和开销3个指标进行了技术评价.将所提混淆方法用于6个恶意软件触发条件的混淆并对混淆之后的恶意代码进行逆向分析实验,结果表明该混淆方法能够较好地抵抗基于静态分析和符号执行的逆向分析方法.

收稿日期:2014-07-07;**修回日期**:2014-10-11
基金项目:国家"九七三"重点基础研究发展计划基金项目(2013CB834204);国家自然科学基金项目(61300242, 61272423);天津市自然科学基金项目(14JCYBJC15300);中央高校基本科研业务费专项资金项目(65121012)
通信作者:王 志(zwang@nankai.edu.cn)

关键词　代码混淆;控制流混淆;代码移动;逆向工程;符号执行

中图法分类号　TP309

全球每天会出现大量新的恶意代码[1],理解恶意代码内部逻辑和恶意行为及其触发条件是恶意代码分析的工作内容.在大部分恶意代码中,恶意行为的触发取决于特定的输入,由条件跳转指令判断其是否执行.在运行时,条件跳转指令会暴露恶意代码的路径信息,例如分支条件、分支的目标地址等.如图1所示,这些暴露的路径信息使得逆向分析者可以通过执行轨迹和反汇编代码探测程序内部逻辑,从而推断出恶意行为的触发条件.

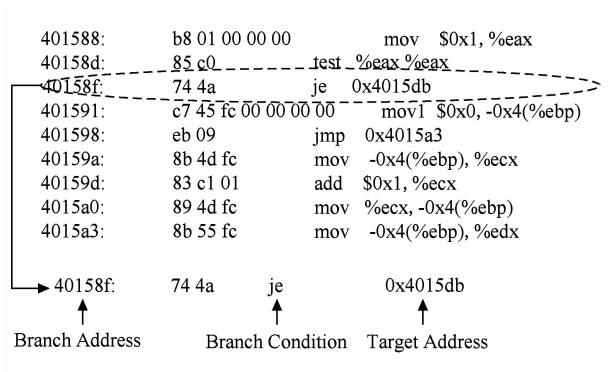


Fig. 1　An example for path information leakage.

图1　路径信息泄漏

近年来,分析者们提出了一些强大的方法来分析恶意代码.一般情况下,分析者可以获取程序全部的二进制代码,将待分析的二进制代码置于可控的虚拟环境中,通过虚拟环境中的CPU得到其执行轨迹,利用条件跳转指令泄漏路径约束信息,使用符号执行技术从执行轨迹中收集谓词逻辑,继而通过约束求解准确地推断出程序的内部逻辑,过程如图2所示:

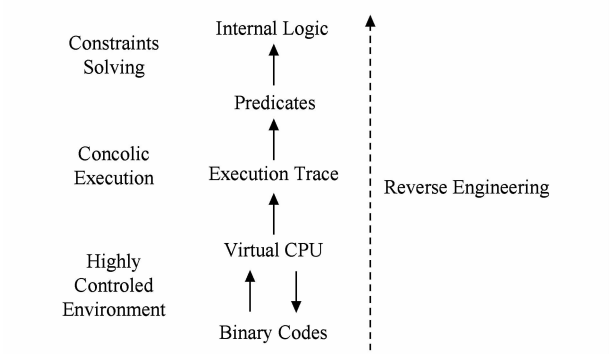


Fig. 2　Reverse engineering.

图2　逆向工程

随着逆向分析技术的发展,恶意代码的编写者不断翻新代码混淆技术来对抗逆向分析,由于完美的程序混淆器是不存在的[2],因此需要科研工作者提出更好的程序保护方法.在已有的工作中,代码混淆的研究主要集中于对抗静态反汇编和反编译,控制流混淆主要集中于目标地址的混淆,通过引入基于函数或变量的间接跳转混淆目标地址以对抗静态分析.但是,当前的逆向工具可以准确地收集软件执行轨迹中暴露的路径信息,很大程度上克服了混淆目标地址带来的分析困难.

综合前人的经验和不足,本文的工作集中于更好地对抗静态反汇编和符号执行.为此,本文对程序的控制流进行动态分割增加静态反汇编的复杂度,相比于混淆无条件跳转指令(例如jmp,call等),本文提出的控制流混淆方法可以同时混淆恶意代码控制流的路径分支条件和目标地址.引入代码移动技术来造成本地程序控制流信息的部分缺失,路径分支结构的分支条件和目标地址均存储于恶意代码分析者不可控制的远程可信实体中,被隐藏的控制流信息呈现良好的黑盒特性.分析者无法获取程序全部的二进制代码,其行为与结构无法预测,从而防范了路径信息的收集,抵抗了基于路径敏感分析的逆向工程,增大了推理恶意行为触发条件的难度并隐藏了恶意行为的入口地址.

本文的主要贡献在于:1)引入代码移动技术用于隐藏程序重要的控制逻辑代码.2)扩展了基于代码移动的代码混淆方法[3],使其良好地应用于二进制程序(包括Linux系统的ELF格式程序和Windows系统的EXE格式程序)控制流混淆,对无条件控制流转移和有条件的控制流转移均可良好地使用.3)证明了该混淆方法的安全性.

1　相关工作

符号执行是近年来涌现出的重要的代码分析技术之一.符号执行技术通常结合了污点分析和定理证明,已经成为软件分析领域中一个强大而有效的手段.利用符号执行进行自动地测试可以遍历程序大部分路径[4-7].Brumley等人[8]推出了Mine-Sweeper工具,该工具综合了静态分析和符号执行技术,用于检测触发型恶意代码的触发条件和恶意行为.2009年,Caballero等人[9]利用符号执行和约束求解技术,从二进制代码中提取某些功能的相关代码和接口

模式,将其应用到其他程序的开发应用中,实现了二进制代码的复用.2012 年,王志等人[10]提出了用以对抗环境敏感技术的恶意代码脱壳方法,利用程序执行时的路径信息泄漏,通过动态污点分析定位程序的脱壳路径并清除恶意代码的环境敏感信息.2013 年 Zeng 等人[11]通过动态符号执行与污点分析技术将被混淆二进制代码转化为更易于理解的高级语言并对其进行动态拷贝盗用.

如此高效的逆向分析技术均是利用软件动态执行中泄漏的路径信息(主要为控制流中条件跳转语句)推理出程序的内部逻辑关系,如何提出更好地保护软件内部逻辑关系的方法是科研工作者应该继续关注研究的.

为了对抗诸如符号执行技术等新型逆向分析技术,2008 年,Sharif 等人[12]提出了通过使用单向杂凑函数加密程序判断逻辑代码来对抗符号执行与约束求解.但是经杂凑函数映射后产生的关键字不具备保序性,即当 $x<c$ 时不意味 $\mathrm{Hash}(x)<\mathrm{Hash}(c)$ 同时成立,该方法仅可混淆判断条件为等于关系的路径分支,有较大的适用范围局限性.2011 年,Wang 等人[13]提出了利用未解数学猜想 $3x+1$ 的非加密的代码混淆方法,该方法利用符号执行无法良好处理循环结构的缺陷和 $3x+1$ 数列的收敛性迷惑定理证明工具,使得路径约束难以表示为一个确定的布尔表达式,从而增加了定理证明工具的推理难度.但是其保护强度受原始路径分支条件的限制,当路径分支条件较为宽泛时,逆向推理的难度较小且同样存在只能用于等于关系判断的缺陷.贾春福等人[14]提出了基于 CPU 指令副作用的路径混淆策略,使用 CPU 指令的副作用隐藏路径分支信息.2014 年王怀军等人[15]发展了基于变形的二进制代码混淆技术,通过对 PE 文件的分片乱序对抗逆向分析技术.在以上方法中,攻击者可以获取程序的全部代码,则攻击者仍可在一定时间内消除混淆对理解程序内部逻辑带来的影响.

2 基于代码移动的分支混淆方法

2.1 基本思想

文献[3]指出,在分析者能够获取恶意程序全部代码的前提下,现今的大部分二进制代码混淆技术是不够健壮的.由于逆向分析者有足够的知识、时间和逆向分析工具,即使程序存在混淆的代码,他们也能在一定的时间内还原被混淆的代码.Falcarin 等人[3]和 Ceccato 等人[16]于 2011 年将代码移动技术应用于软件保护领域,提出了基于代码移动技术的二进制代码混淆方法.将程序重要二进制代码通过代码移动技术移动至分析者无法篡改和入侵的可信实体,可以限制分析者控制恶意程序的全部二进制代码,使得程序的行为不可预测.在运行时,原程序中被移动的代码会在远程可信实体中执行.但是代码分割(如整块代码移动等)会使得本地代码与远程可信实体频繁交互而导致较大的网络开销.Wang 等人[17]于 2012 年提出了基于代码移动的控制流混淆方法,将程序路径分支判断条件隐藏于远程可信实体中.但是由于路径选择在本地代码中完成,所以在二进制代码中会因无法隐藏分支的目标地址而暴露 2 条分支路径.

本文着力于解决现有的基于代码移动的混淆方法中存在的交互过频与混淆强度问题,提出了一种基于代码移动的二进制代码控制流混淆方法.该方法动态分割了程序的控制流,将程序重要的路径控制逻辑代码移动至逆向分析者无法触及的远程可信实体,相对于前人的方法,该方法相对有效地解决了频繁交互带来的执行效率低下的问题并进一步增加了安全性.使用代码移动方式进行二进制代码控制流混淆有如下优势:1)程序在不可信的主机运行时,经混淆后的本地程序二进制代码不完整,代码移动可以保护程序来对抗静态和动态分析;2)在每次运行时,程序的行为完全取决于代码编写者在远程可信实体设定的信息,这样使得程序的行为无法预测;3)对控制逻辑代码采取保留语义的整体移动方式,因此对任何分支判断条件均可使用,解决了某些代码混淆方式应用范围的限制.

程序的路径分支可以作如下形式化表示:设程序的某个路径分支结构(二进制代码的输出只有 2 个分支:真分支(true branch)和假分支(false branch)可能的输入值的集合为 $I=\{I_1,I_2,I_3\cdots\}$,对应的输出行为集合 $O=\{O_{\mathrm{Ture}},O_{\mathrm{False}}\}$,相应的输出行为代码块的入口地址集合为 $S=\{S_{\mathrm{True}},S_{\mathrm{False}}\}$,分支判断条件可视为输入集合与输出集合的映射 δ.本文提出思想重点在于同时混淆输入输出的映射关系 δ 和输出行为代码块的入口地址.在源码中将输入与输出的映射关系 δ 使用混淆函数 f 替代,映射关系 δ 和输出行为代码块的入口地址集合 S 由远程可信实体予以保护.本地程序通过与可信实体的交互来获取本地代码中缺失的控制流信息.

2.2　实现方式

为了更好地阐述该混淆方法的基本思想,本文将在此节表述该混淆方法的实现方式.图3给出了混淆模型,虚线表示的是分支的执行过程,实线部分是程序被混淆后的执行过程.

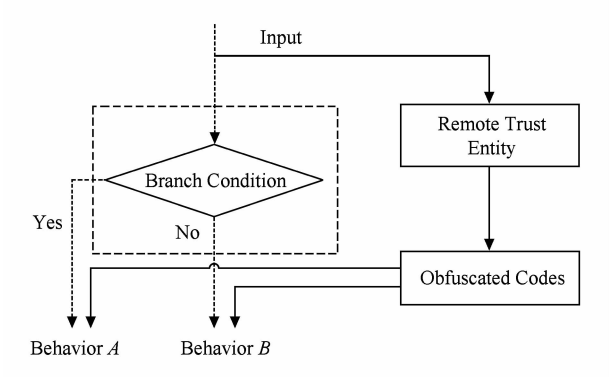


Fig. 3　A framework of obfuscation.

图3　混淆模型

1) 分支判断条件的隐藏

P 是一个运行在不可信环境中被混淆的程序,其中包含替代原程序中相应分支判断条件的函数 f.本实验利用了远程可信实体,该可信实体可以为网络中的1台计算机或设备,逆向分析者无法篡改其数据甚至无法知道此可信实体何时启动.被转移的代码在可信实体中执行,恶意代码可以与可信实体进行动态地交互来得到当前输入与输出的映射关系.可信实体与程序 P 的通信交互由函数 f 执行,函数 f 使用 socket 编写,其主要功能为完成对被混淆路径分支输入和远程可信实体发回数据的处理,而且其构造了2个逻辑信道:请求信道用于向可信实体发送当前的输入值,响应信道用于接收可信实体发回的分支判断结果,其模型如图4所示:

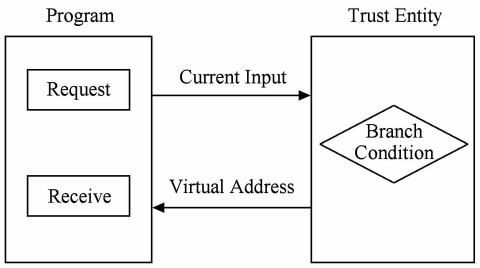


Fig. 4　Hidden branch condition.

图4　隐藏分支判断条件

例如,以简单的 if 条件分支结构为例:

if $x==5$

　Behavior A;

else

　Behavior B;

endif

本文假设恶意代码 P 中某一条路径分支判断条件为 $x=5$,即 x 为一输入变量,操作数5为触发条件,输入等于5时,执行行为 A,反之则执行行为 B.在源代码中,使用混淆函数 f 向可信实体发送当前的输入值并替代原条件判断语句,原条件判断语句以相同的语义放置于远程可信实体中,程序运行时变量 x 被发送至可信实体,条件判断会在可信实体中执行,因此在本地代码中无法获取路径分支条件.

2) 目标地址的隐藏

在移动重要的控制逻辑代码后,还需在本地的程序中混淆原路径分支处产生的2条分支入口地址.由于被混淆的分支判断条件在本地程序中被移除,路径分支选择功能需要添加混淆代码去动态地还原.在混淆前的代码中,路径分支条件 $x=5$ 会根据不同的输入产生 A 与 B 这2种行为.源代码被混淆后编译为可执行文件,编译器会赋予 A 和 B 这2种行为相应的文件偏移地址(file offset),在程序装入内存运行时,系统会将文件偏移映射为虚拟地址(virtual address),在通常情况下,编译好的二进制文件中各指令的文件偏移地址和虚拟地址是不会产生变化的.将两行为入口的虚拟地址通过反汇编找出并存储于远程可信实体中.恶意程序在被感染机器执行至原分支结构时启动函数 f,将输入发送至远程可信实体,可信实体将得到的输入与其相应的映射关系比对,向本地程序发回相应输出的内存地址,混淆代码完成相应的处理后进行跳转.下面通过2个具体的实例,表明如何通过插入混淆代码隐藏分支行为的目标地址.

以2.2节的条件分支为例,在 Windows 操作系统中,混淆代码为

```
recvaddr=obfuscation(x);
_asm{
    mov eax, recvaddr;
    jmp eax;
    }
```

在 Linux 操作系统中,混淆代码为

```
recvaddr=obfuscation(x);
_asm__volatile_("movl%0,%%eax \n\t"
      "jmp * %%eax\n\t"
  :
  :"r"(recvaddr)
  :"%eax");
```

该混淆代码有如下功能:1)替换条件跳转指令,静态地隐藏路径分支的目标地址;2)处理选定的寄存器(本文设定为 *eax*)来存放接收到的虚拟地址数据;3)完成跳转,动态地还原被破坏的控制流.在远程可信实体没有向本地程序发送数据时,指定的寄存器中所包含的操作数是随机的,不包含任何的语义.

3 技术评价

3.1 评价标准

Collberg 等人[18]对代码混淆技术提出了 3 个评价指标:强度(potency)、弹性(resilience)和开销(cost).本节将依据 Collberg 等人提出的评价指标,结合静态分析和符号执行的特点,对本文提出的方法进行定性评价.

强度为使用混淆算法后,逆向分析人员理解程序难度,在本文中可解释为混淆算法对程序控制流信息的隐蔽程度.P 和 P' 分别表示未经混淆的原程序和混淆后的程序.隐蔽信息包括分支点 $H_t(p')$、分支条件 $H_c(p')$和分支目标地址 $H_d(p')$,其强度可用隐藏控制信息的比率表示

$$T_{po}=\frac{H_t(p')+H_c(p')+H_d(p')}{H(p)}\times 100\%.$$

弹性为混淆算法抵御攻击和逆向分析的能力,其包括逆向分析人员在推理可信实体隐藏的控制逻辑代码时的难度 C_{rea}和恢复控制流的难度 C_{rec}.代码移动技术的弹性为

$$T_{re}=Resilience(C_{rea},C_{rec}).$$

开销为代码混淆给程序带来的额外开销,它包括时间开销 C_{time}和空间开销 C_{size}两部分.代码移动的开销为

$$T_{co}=Cost(C_{time},C_{size}).$$

3.2 基于代码移动的混淆技术评价

1) 强度与弹性

① 强度.对于本文提出的混淆方案对原程序控制流进行动态分割,代码混淆的强度根据隐蔽控制信息的多寡加以判别.本文对程序路径分支处的分支条件和输出行为的目标地址同时予以隐蔽,条件跳转指令被混淆为无条件跳转指令,包含地址信息的寄存器初始操作数无任何意义.如果攻击者逆向分析被混淆的程序,只能通过动态分析得到触发行为与函数 f 的一个返回值的关系,由于可信实体隐蔽了相应的控制逻辑代码,攻击者无法获得更多的程序路径信息,无法判断被混淆处是否为路径分支或无法预测会产生多少条路径分支,也无法定位所有路径分支在内存中的虚拟地址.因此,本文提出的代码混淆方法有较好的强度.

② 弹性.从逆向分析角度去分析被混淆的程序,将其反汇编后如图 6 所示,若要定位 jmp eax 指令具体的跳转目标地址,必须分析位于虚拟地址 0x401000 的函数,也就是插入的混淆函数 f、远程可信实体保护了程序重要的控制逻辑代码,被混淆处的代码可以看作为近似黑盒的.对于一个近似黑盒的程序,想要理解其内部功能唯一能做的只有对其进行 Oracle Access[19],即给定有限多个输入观察其输出结果.由于本地代码的不完整性使得程序行为无法预测,为被混淆的程序设计足够多的输入是非常困难的.因此,本文提出的代码混淆方法有较强的弹性.

2) 开销

在时间开销中,恶意代码混淆后由于单个混淆函数 f 的时间复杂度为 $O(1)$,函数 f 代替了原程序的控制信息,未改变衡量原程序时间复杂度的多项式,混淆后的程序时间复杂度仍服从于原程序的时间复杂度;同时由于与可信实体通信次数和每次通信的数据量有限,即使受制于带宽的影响,传输增加的时间开销也在可以容忍的范围内.在空间开销中,空间开销为程序占用的存储空间和程序指令数的大小.程序插入的混淆函数 f 为简单的 socket 函数,混淆代码为简短的汇编指令(如 2.2 节中的代码所示),因此代码移动方法对程序的空间开销影响较小.

4 实验结果

4.1 对抗静态分析

以 2.2 节中的 if 分支语句为例,通过 IDA 静态分析工具生成的控制流图,其混淆前后的控制流图如图 5、图 6 所示.

在图 5 中,IDA 静态分析工具可以分析出该路径分支的跳转条件,并能收集到其产生的 2 条跳转路径.图 6 可见,经过使用本文方法混淆后的路径分支处会产生控制流的断流,当程序运行时远程可信实体返回相应的目标地址,控制流才可恢复.在静态控制流图中,混淆处由混淆代码中的无条件跳转指令替代了暴露程序路径分支信息的条件跳转指令,指定寄存器在静态分析中无任何语义,静态分析无法收集相应跳转条件和跳转产生的所有路径,因此该代码混淆方法较好地对抗了基于反汇编的静态分析.

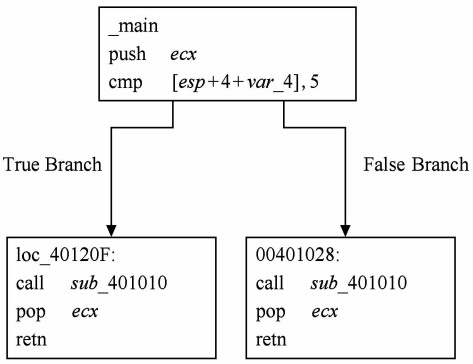


Fig. 5　Control flow graph before obfuscation.

图 5　混淆前控制流图

```
_main:
push  ebp
mov   ebp,esp
sub   esp,8
push  ebx
push  esi
push  edi
mov   eax,[ebp+var_8]
push  __
call  sub_401000
add   esp,4
mov   [ebp+var_4], eax
mov   eax,[ebp+var_4]
jmp   eax
```

Fig. 6　Control flow graph after obfuscation.

图 6　混淆后控制流图

4.2　对抗符号执行与约束求解

在对抗符号执行方面,我们使用最近较为流行的自动分析工具 BitBlaze[20] 对被混淆的恶意代码进行分析和生成执行轨迹,在可信实体未启动的情况下 BitBlaze 无法生成正确的执行轨迹. 如表 1 可见,通过静态分析模块对程序的条件跳转数进行统计,被混淆后的恶意代码中条件跳转数较混淆前有所减少(不统计混淆函数所引入的条件跳转),本地代码的不完整性令 BitBlaze 的执行轨迹无法遍历所有的路径空间.

Table 1　Number of Conditional Jump

表 1　条件跳转指令数

Program	Origin	Obfuaction	Cost
Win32. Worm. Blaster	1 361	1 360	−1
Win32. Malware. ssc	8	5	−3
Win32. Worm. Darker	17	14	−3
Linux/Sickabs!!!	21	16	−5
Linux/ELFPatch. B	38	31	−7
Linux/ELF. Manpages	11	9	−2

Note: Negative cost indicates conditional jump is hidden after being obfuscated.

STP[21] (simple theorem prover)是一款约束求解工具,其可以识别出 C/C++语言或机器语言中除浮点数运算外的几乎所有函数与谓词结构. 实验表明,利用 STP 对程序的执行轨迹约束求解,本文提出的混淆方法在恶意代码中插入了非条件跳转指令代替了条件跳转指令,本地恶意程序的二进制代码中相应位置不存在谓词逻辑,所以 STP 无法收集相应的路径分支信息和推理恶意行为路径的可达性和执行条件,达到了较好对抗符号执行与约束求解的目的.

4.3　程序的开销

为不失一般性,本文选取了 Linux 平台和 Windows 平台下的 6 种恶意代码为例,对程序的开销进行讨论. 在 Windows XP SP3 操作系统,选用 VC++ 6.0 为编译工具,Debug 编译模式下对被混淆的源代码进行编译;在 Ubuntu 13.04 操作系统,使用 GCC 为编译工具,无优化编译. 程序运行的硬件环境为 Intel Core2 Q9400 CPU,2 GB RAM,可信实体为阿里云中的虚拟机,CPU:1 核,操作系统:Windows Server 2003 R2 标准版 SP2 32 位中文版,内存:1 GB,地域:青岛,网络带宽:1 Mbps. 为了令程序能正常执行,网络连接协议选用 TCP 协议保证通信过程的完整. 通过反编译工具对程序混淆前后占用的空间和指令数进行了统计,其结果见表 2 和表 3 所示. 同时,选取了 Sartaj Sahni 编写的《数据结构算法与应用——C++语言描述》中顺序查找、二分

Table 2　The Cost in Size

表 2　空间开销　　B

Program	Origin	Obfuaction	Cost
Win32. Worm. Blaster	180 335	200 810	20 475
Win32. Malware. ssc	184 320	184 320	0
Win32. Worm. Darker	561 152	561 152	0
Linux/Sickabs!!!	12 178	12 363	185
Linux/ELFPatch. B	11 953	12 138	185
Linux/ELF. Manpages	12 148	12 445	297

Table 3　The Cost in Instructions

表 3　指令数开销

Program	Origin	Obfuaction	Cost
Win32. Worm. Blaster	39 130	44 141	5 011
Win32. Malware. ssc	37 274	37 924	650
Win32. Worm. Darker	162 396	162 674	278
Linux/Sickabs!!!	737	870	133
Linux/ELFPatch. B	1 176	1 306	130
Linux/ELF. Manpages	675	879	204

查找和 Hash 查找这 3 种查找方式的源代码,设定 1 000 个伪随机数,在最坏情况下测量本文提出的代码混淆方法对不同时间复杂度程序时间开销的影响,利用 wireshark 软件对本地程序与可信实体的通信进行抓包,通过网络流量分析网络开销对程序整体额外开销的影响,结果如表 4 所示:

Table 4 The Time Cost and Network Cost

表 4 时间开销和网络开销

Program	Origin	Obfuaction	Network Transit Time/s	Number of Package	Total Package Size/B
Sequential Search	0.000 084	144.163 329	143.984	7 000	510 000
Binary Chop Search	0.000 059	1.488 128	1.334 596	70	5 100
Hash Search	0.000 038	0.110 10	0.100 23	7	510

由表 2 数据可见,本文提出的混淆方法对程序的空间开销影响较小,由于程序编译时需要进行区块对齐,在一定情况下不会增加程序的文件体积. 表 3 中的数据表示了本文提出的混淆方法对程序的指令数影响较小. 表 4 中的数据说明,随着原程序时间复杂度的增加,相应的额外开销也成正相关增加,在程序实际运行时,时间开销绝大部分来源于网络传输过程,平均每次网络传输的数据量很小,只有 73 B. 但是顺序查找在最坏情况的时间复杂度为 $O(n)$,需要执行 1 000 次查找操作,在该种情况下网络传输的数据量会显著增加.

5 结 论

针对程序路径信息泄漏问题和传统代码混淆方法强健性与执行效率问题,本文提出了一种基于代码移动的二进制程序控制流混淆方法,它将程序的控制逻辑代码隐藏于无法被逆向分析者控制的远程可信实体,通过插入包含无初始意义操作数的非条件跳转指令隐藏路径分支的目标地址,逆向分析者无法获取程序全部的二进制代码,增大了推理和预测程序行为的难度. 同时,还对本文提出的代码混淆方法的强度、弹性和开销进行了定性评价. 实验表明,该方法良好地抵抗了静态和动态分析.

相比于前人提出的基于代码移动的代码混淆方法,本文提出的代码混淆方法相对地解决了文献[3]方法中整体代码块移动而导致的大量网络数据传输问题,同时解决了文献[17]方法中目标地址暴露的问题,具有相对更好的安全性,相比于文献[12]的方法具有更大的适用范围.

时间开销大量分布于网络传输开销,单次通信的数据量较少,对于恶意代码的混淆,本文提出的方法具有较好的应用价值. 但是在从软件保护的角度来看,本文提出的混淆方法应用在高时间复杂度的算法(如循环、嵌套循环等)时会频繁地与可信实体进行交互,其应用于高时间复杂度算法的方法还需进一步优化.

在未来的工作中,我们将通过加密网络传输数据等手段对本地程序与可信实体交互的过程进行保护;对程序的控制流分割进行优化,通过减少与可信实体交互次数来减少被混淆程序的时间开销,使其更好地适用于软件保护的要求;对更多的攻击手段进行评估分析,利用可信实体接收到的数据特征判断程序是否处于虚拟的调试环境,更好地减少路径信息泄漏.

参 考 文 献

[1] Symantac. 2015 Internet Security Threat Report [OL]. [2014-07-06]. https://know.elq.symantec.com/LP=1542

[2] Barak B, Goldreich O, Impagliazzo R, et al. On the (im) possibility of obfuscating programs [C] //Proc of Int Cryptology Conf(CRYPTO 2001). Berlin: Springer, 2001: 1-18

[3] Falcarin P, Carlo S D, Cabutto A, et al. Exploiting code mobility for dynamic binary obfuscation [C] //Proc of Internet Security (WorldCIS). Piscataway, NJ: IEEE, 2011: 114-120

[4] Godefroid P, Levin M Y, Molnar D A. Automated whitebox fuzz testing [C] //Proc of the 16th Network and Distributed System Security Symp. Piscataway, NJ: IEEE, 2008: 151-166

[5] Cadar C, Sen K. Symbolic execution for software testing: Three decades later [J]. Communications of the ACM, 2013, 56(2): 82-90

[6] Farooqui N, Schwan K, Yalamanchili S. Efficient instrumentation of GPGPU applications using information flow analysis and symbolic execution [C] //Proc of the 7th Workshop on General Purpose Processing Using GPUs. New York: ACM, 2014: 19-27

[7] Bugrara S, Engler D. Redundant state detection for dynamic symbolic execution [C] //Proc of the 21st USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2013: 199-211

[8] Brumley D, Hartwig C, Liang Z, et al. Automatically identifying trigger-based behavior in malware, CMU-CS-07-105 [R]. Pittsbutgh, PA: Carnegie Mellon University, 2007

[9] Caballero J, Johnson N M, McCamant S, et al. Binary code extraction and interface identification for security applications, UCB/EECS-2009-133 [R]. Berkeley, CA: Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2009

[10] Wang Zhi, Jia Chunfu, Lu Kai. Malicious hidden-code extracting based on environment-sensitive analysis [J]. Chinese Journal of Computers, 2012, 35(4): 693-702 (in Chinese)
(王志,贾春福,鲁凯. 基于环境敏感分析的恶意代码脱壳方法[J]. 计算机学报, 2012, 35(4): 693-702)

[11] Zeng J, Fu Y, Miller K A, et al. Obfuscation resilient binary code reuse through trace-oriented programming [C] //Proc of the 20th ACM SIGSAC Conf on Computer & Communications Security. New York: ACM, 2013: 487-498

[12] Sharif M I, Lanzi A, Giffin J T, et al. Impeding malware analysis using conditional code obfuscation [C] //Proc of the 16th Network and Distributed System Security Symp. Piscataway, NJ: IEEE, 2008: 321-333

[13] Wang Z, Ming J, Jia C, et al. Linear obfuscation to combat symbolic execution [C] //Proc of the 16th Computer Security-European Symp on Research in Computer Security. Berlin: Springer, 2011: 210-226

[14] Jia Chunfu, Wang Zhi, Liu Xin, et al. Branch obfuscation: An efficient binary code obfuscation to impede symbolic execution [J]. Journal of Computer Research and Development, 2012, 48(11): 2111-2119 (in Chinese)
(贾春福,王志,刘昕,等. 路径模糊:一种有效抵抗符号执行的二进制混淆技术[J]. 计算机研究与发展, 2012, 48(11): 2111-2119)

[15] Wang Huaijun, Fang Dingyi, Li Guanghui, et al. Research on deformation based binary code obfuscation technology [J]. Journal of Sichuan University: Engineering Science Edition, 2014, 46(1): 14-21 (in Chinese)
(王怀军,房鼎益,李光辉,等. 基于变形的二进制代码混淆技术研究[J]. 四川大学学报:工程科学版, 2014, 46(1): 14-21)

[16] Ceccato M, Tonella P. Codebender: Remote software protection using orthogonal replacement [J]. Software, 2011, 28(2): 28-34

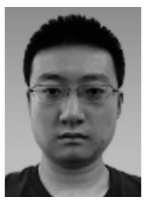
[17] Wang Z, Jia C, Liu M, et al. Branch obfuscation using code mobility and signal [C] //Proc of the 7th IEEE Int Workshop on Security, Trust, and Privacy for Software Applications. Piscataway, NJ: IEEE, 2012: 553-558

[18] Collberg C, Thomborson C, Low D. A taxonomy of obfuscating transformations, 148 (1997) [R]. Auckland, New Zealand: Department of Computer Science, University of Auckland, 1997

[19] Beaucamps P, Filiol É. On the possibility of practically obfuscating programs towards a unified perspective of code protection [J]. Journal in Computer Virology, 2007, 3(1): 3-21

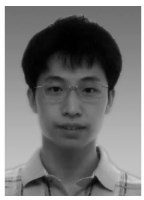
[20] Song D, Brumley D, Yin H, et al. BitBlaze: A new approach to computer security via binary analysis [C] //Proc of 2008 Int Conf on Information Systems Security. Berlin: Springer, 2008: 1-25

[21] Ganesh V, Dill D L. A decision procedure for bit-vectors and arrays [C] //Proc of the 19th Int Conf on Computer-Aided Verification. Berlin: Springer, 2007: 519-531



Chen Zhe, born in 1989. MSc candidate. His main research interests include code obfuscation and software watermarking.

Wang Zhi, born in 1981. PhD, lecturer. His main research interests include code obfuscation, malware analysis and prevention.



Wang Xiaochu, born in 1989. MSc candidate. His main research interests include code obfuscation.

Jia Chunfu, born in 1967. PhD, professor and PhD supervisor. His main research interests include network and system security, cryptography application and malware analysis.