

基于融合编译的软件多样化保护方法

熊小兵, 舒辉, 康绯

(信息工程大学, 河南 郑州 450001)

摘要: 针对现有主流保护方法存在的特征明显、模式单一等问题, 以 LLVM 开源编译框架为基础, 提出了一种基于融合编译的软件多样化保护方法, 该方法将目标软件进行随机化加密处理, 并在编译层面与掩护软件进行深度融合, 通过内存执行技术, 将加密后的目标软件进行解密处理, 进而在内存中以无进程的形式执行, 利用掩护代码的多样性、融合策略的随机性来实现目标软件的多样化保护效果。选取了多款常用软件作为测试集, 从资源开销、保护效果、对比实验等多个角度对所提方法进行了实例测试, 从测试结果可以看出, 所提方法资源开销较小, 相较于混淆、加壳等传统方法, 所提方法在抗静态分析、抗动态调试等方面具有较大优势, 能有效对抗主流代码逆向分析和破解手段。

关键词: 软件保护; 多样化; 融合编译; 内存执行; 底层虚拟机; 中间表示

中图分类号: TP393

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2020075

Method of diversity software protection based on fusion compilation

XIONG Xiaobing, SHU Hui, KANG Fei

Information Engineering University, Zhengzhou 450001, China

Abstract: For the obvious characteristics and single mode of the existing common protection methods, with the help of the LLVM framework, a diversity software protection method based on fusion compilation was proposed. In the proposed method, the target software is encrypted randomly, and deeply integrated with the bunker code at the compilation level, and the encrypted target software is decrypted by memory execution technology. Then it is executed in the form of no process in memory, and the diversified protection effect of the target software is realized by the diversity of the bunker and the randomness of the fusion strategies. A number of commonly used software are selected as the test case, and the proposed method is tested from the aspects of resource cost, protection effect, comparative experiment and so on. Compared with the traditional methods such as obfuscation and packing, the

收稿日期: 2019-07-21; 修回日期: 2019-09-27

通信作者: 熊小兵, bingxiaoxiong@163.com

基金项目: 国家重点研发计划 (2016YFB08011601)

Foundation Item: National Key Research and Development Project (2016YFB08011601)

论文引用格式: 熊小兵, 舒辉, 康绯. 基于融合编译的软件多样化保护方法[J]. 网络与信息安全学报, 2020, 6(6): 13-24.

XIONG X B, SHU H, KANG F. Method of diversity software protection based on Fusion compilation[J]. Chinese Journal of Network and Information Security, 2020, 6(6): 13-24.

proposed method has great advantages in anti-static analysis and anti-dynamic debugging, and can effectively resist the mainstream methods of reverse analyzing and cracking.

Key words: software protection, diversification, fusion compilation, memory execution, LLVM, intermediate representation

1 引言

软件是互联网中各类信息系统的核心组成部分,与实体商品不同,软件容易被复制、篡改、破解和传播,造成互联网上存在大量盗版软件,互联网用户多在不知情的情况下对盗版软件进行下载或传播^[1],这给软件开发者的合法权益造成了巨大损失,在很大程度上降低了软件开发者的积极性,不利于互联网生态的良性健康发展^[2]。如何对开发的软件代码进行有效保护,是互联网发展过程中一直存在的 key 问题,特别是在软件逆向分析技术飞速发展的严峻形势下,大力发展软件保护技术显得尤为重要和迫切^[3-4],亟需研究先进的软件保护技术和方法,来提高软件的抗逆向、抗破解能力。

2 常用软件保护技术

软件一般以二进制文件形态发布,软件盗版本在无法准确获取软件源码的情况下,常采用软件逆向方法对软件的实现过程进行分析,找出破解点,去除软件注册码、软件加密等保护机制,达到软件破解目的。常用的软件逆向手段有静态逆向分析和动态调试分析。静态逆向分析主要是指利用 IDA 等反汇编、反编译软件,提取出目标软件的汇编代码,甚至是源码,分析出软件的实现机制和关键代码,进而在二进制层面对其进行篡改破解。动态调试分析是指利用 OllyDBG、WinDBG 等动态调试工具,对软件的动态执行过程进行指令级跟踪分析,定位关键代码段,进而对其进行篡改破解。

为了防止合法软件被盗版,开发者一般会采用软件保护技术对软件进行处理,主要包括基于硬件和基于软件的保护方式。软件开发者对软件保护技术有一个典型误区,认为无法被破解的软件保护技术才是可行的。实际上,根本不存在无法破解的软件,无论采用何种软件保护技术,软件都有被破解的风险。所有软件保护技术的核心

都是尽可能加大软件被破解的难度(即增大软件的抗逆向破解能力),使逆向分析人员难以在短时间内分析出软件工作机理和组织结构。目前,较为常用的软件保护技术有加壳保护、代码混淆保护等。

2.1 软件加壳保护

大自然中植物的壳用来保护种子,计算机软件的壳是一段专门用来保护软件不被非法篡改或反编译的代码或程序^[5]。加壳是软件保护的常用手段,一般利用特殊的算法,对 EXE、DLL 或 OCX 程序中的资源进行压缩和加密,类似压缩软件的效果,只不过被压缩后的程序,仍然可以独立运行,在内存中完成解密和解压缩操作^[5]。加壳程序的原始代码在文件中一般被加密处理,无法查看源代码,运行后在内存中还原,这样就可以有效地防止要保护的程序被静态反编译,同时可以防止原始程序被非法修改^[6]。

加壳后程序运行,壳会附加在原始程序上,通过 Windows 加载器载入内存,先于原始程序得到执行控制权^[7]。壳程序在执行过程中对原始程序进行解密、解压缩、还原操作,而后把控制权交还给原始程序。

软件壳一般分为压缩壳、加密壳和虚拟机壳 3 种。压缩壳仅对原始程序中的资源进行压缩,如 Aspack、UPX 和 PeCompact 等,使用压缩壳保护的程序,其大小一般小于原始程序。加密壳不仅对原始程序中的资源进行压缩,还具备函数输入表加密、反调试、双进程保护和虚拟执行等功能,如 ASProtect、Armadillo 和 Themida 等^[8]。虚拟机壳一般将 PE 文件反汇编为 X86 指令流,并从中提取目标指令序列,而后将目标汇编指令流转换成字节码,PE 文件中被保护代码的正确运行需要虚拟机解释器对字节码解释执行^[9-11],虚拟机壳自带一个虚拟执行引擎,会对程序执行效率带来较大影响。

2.2 代码混淆保护

代码混淆是一种常用的软件保护方法,混淆

前后的程序在功能和行为上保持一致, 主要目的是让逆向分析者难以运用逆向工具提取出软件关键代码。代码混淆的系统性研究是从 20 世纪 90 年代末开始的, 其基础性工作是由 Collberg、Thomborson、Low 等完成的^[12-13]。Collberg 给出了代码混淆的定义, 并第一次对代码混淆进行了分类, 详细总结了各类代码混淆技术, 提出对代码混淆评价的性能指标。根据混淆原理和对象的不同, 可将代码混淆技术分为外形混淆、控制混淆、数据混淆和预防混淆^[14-16]。外形混淆是一种针对程序外形及布局的变换来迷惑破解者的混淆, 数据混淆是一种针对数值变量及数据类型变换的混淆, 控制混淆是一种通过改变程序代码的控制流结构来增加代码结构复杂度的混淆方法, 预防混淆是一种针对专用反汇编器或反编译器的混淆。

最初的代码混淆主要应用于高级语言, 即源码层面的代码混淆。通过对源码形态的变换, 生成具有抗逆向功能的可执行程序或脚本程序。常用混淆技术分别在源码、二进制程序和编译中间表示 3 个层次实现。源码级的混淆难度最小, 但需要考虑具体编程语言的语法规则, 在实施变换的过程中有一定的限制; 二进制程序的重构需要建立在对程序正确分析的基础上, 且由于代码结构相对固定, 修改难度较大; 编译中间表示介于两者之间, 格式相对规范, 程序内容信息全面, 修改难度折中且方式灵活。文献[13, 17-19]虽然是对编译器中间表示进行混淆处理, 但选取的混淆对象是中间表示文本文件, 并没有发挥编译器强大的接口功能, 从严格意义上讲, 并不是真正基于融合编译的代码混淆。

2.3 传统软件保护方法存在的问题

基于代码加壳和代码混淆的传统软件保护能在一定程度上解决软件保护问题, 但这两类方法存在明显的弊端。

对于软件加壳保护, 很多通用壳具有明显的结构特征, 逆向分析人员可以快速提取出加壳后的代码特征, 识别出壳类型, 针对某种特定的壳, 利用专用脱壳工具, 可对加壳后的代码进行快速脱壳处理, 进而还原出被保护代码。对于虚拟机这类高级软件壳, 脱壳难度较大, 但由于虚拟机

壳是对目标代码逐指令解释执行, 严重影响了程序的执行效率, 加壳过程中额外增加的引擎, 会引起加壳后代码的膨胀问题。

对于基于代码混淆的软件保护技术, 一般针对有源码目标软件, 绝大多数混淆算法无法适用于二进制代码。代码混淆一般只是对指令进行等价替换, 或插入垃圾指令, 或对程序控制流结构进行混淆处理, 混淆前后代码功能基本一致。代码混淆操作对于静态逆向分析具有较好的软件保护效果, 但对于基于动态调试的软件逆向分析手段, 保护效果则非常有限。

综上, 软件逆向分析和破解技术发展迅速, 当前代码加壳、代码混淆等传统的软件保护方法难以对抗日趋先进的破解技术, 软件保护效果并不理想。本文在对传统软件保护方法进行深入分析的基础上, 提出了一种基于融合编译的程序多样化保护方法, 该方法针对传统方法不适用二进制代码保护, 保护后代码特征明显, 易被还原、执行效率低、抗动态调试效果差等典型问题, 主要以无源码的二进制代码为保护对象, 通过编译技术, 将目标代码与随机选择的掩体代码进行深度融合, 利用掩体代码的多样性、目标代码保护策略的随机性, 来实现软件多样化保护效果, 使保护后的代码, 从静态特征、动态行为方面具有很强的多样性特点, 有效对抗主流代码逆向分析和破解手段。

3 软件多样化保护模型和框架设计

3.1 软件多样化保护模型

本文提出的多样化保护方法的核心思想是: 将待保护的目标软件进行加密压缩处理, 与随机选取的掩体程序进行编译层面的深度融合, 在掩体代码执行过程中, 分阶段完成被保护软件的加载执行。本节通过形式化方法, 对多样化保护模型进行详细描述。

定义 1 (二进制程序 P) 二进制程序 P 是由指令、数据, 以及指令对数据的引用关系组成的集合, 记作

$$P = \langle C, D, I \rangle$$

其中, $C = \langle C_1, C_2, \dots, C_n \rangle$, 表示程序中的有序指

令集合; $D = \langle D_1, D_2, \dots, D_m \rangle$, 表示程序中的数据集合, 包括全局数据、常量数据等; $I = \langle I_1, I_2, \dots, I_k \rangle$, 表示程序中的指令对数据的引用关系集合, 包括读、写、调用等引用关系。

定义 2 (代码融合) 代码融合是指将两个程序 P_1, P_2 , 在指令、数据等多个层面进行深度融合, 使其构成一个程序, 但同时具备两个程序的功能, 记作

$$\text{Mix}(P_1, P_2) = (C_{p1}, C_{p2}, D_{p1}, D_{p2}, I_{p1}, I_{p2}, C_{\text{mix}}, D_{\text{mix}}, I_{\text{mix}})$$

其中, C_{mix} 表示用于融合的功能代码集合, D_{mix} 表示用于融合的功能数据集合, I_{mix} 表示用于融合的新增引用关系集合, 这 3 部分共同协作, 完成两个程序的无缝、深度融合。

定义 3 (基于编译融合的代码保护) 基于编译融合的代码保护是指在编译层面, 将目标软件 P_1 以加密方式融合到另一个掩体程序 P_2 中, 生成一个新程序, 实现对目标软件的保护, 目标软件 P_1 整体作为保护后程序的数据部分, 被加密压缩处理, 记作

$$\text{ComMix}(P_1, P_2) = (C_{\text{new}}, D_{\text{new}}, I_{\text{new}})$$

其中,

$$C_{\text{new}} = (C_{p2}, C_{\text{mix}}, C_{\text{decrypt}})$$

$$D_{\text{new}} = (D_{p2}, D_{\text{mix}}, DC_{p1})$$

$$I_{\text{new}} = (I_{p2}, I_{\text{mix}}, I_{\text{decrypt}})$$

保护后程序的代码 C_{new} 由 3 部分组成: 掩体程序 P_2 的代码 C_{p2} 、新加入的融合功能代码 C_{mix} 、解密功能代码 C_{decrypt} 。保护后程序的数据 D_{new} 由 3 部分组成: 掩体程序 P_2 的数据 D_{p2} 、新加入的融合功能数据 D_{mix} 、加密压缩处理后的目标程序 DC_{p1} 。保护后程序的引用关系 I_{new} 由 3 部分组成: 掩体程序 P_2 的引用关系 I_{p2} 、新加入的融合功能引用关系 I_{mix} 、解密功能引用关系 I_{decrypt} 。

解密功能代码是指插入在掩体中的代码片段, 其作用是对加密的目标软件进行解密处理。融合功能代码是指在掩体程序中插入的代码片段, 其核心功能是将目标软件以无进程的形式

在内存中加载执行起来, 实现目标软件和掩体程序的进程空间共享, 完成基于掩体代码的目标软件保护。

定义 4 (程序内存执行) 程序内存执行是指通过自实现的程序加载器, 将目标程序直接在内存中加载运行起来, 实现目标程序的无进程运行。程序内存执行的核心在于自实现的程序加载器 L , 记作

$$L(P) = (L_{\text{map}}, L_{\text{IAT}}, L_{\text{reloc}}, L_{\text{fix}})$$

其中, L_{map} 用于完成目标程序的内存加载, L_{IAT} 用于完成目标程序的导入表地址修复, L_{reloc} 用于完成目标程序的重定位项修复, L_{fix} 用于完成目标程序中 PEB、TLS 等关键数据结构的修复。上述 4 个步骤完成后, 目标软件即可在内存中直接运行起来。

3.2 软件多样化保护框架设计

本文提出的软件多样化保护框架设计如图 1 所示, 输入是待保护的二进制形态目标软件, 输出为多样化保护处理后的软件。框架底层以掩体代码库、加密算法库、编译环境库为基础支撑, 其中掩体代码是指在互联网收集的功能各异的开源代码, 用来掩护目标软件的特征和行为。首先, 利用内存执行技术将目标二进制文件变形成可直接在内存中运行的代码形态, 即在目标二进制文件头部加入一个加载器, 目标软件以数据形式加密后附着在加载器后面; 然后, 利用编译技术将掩体代码编译成中间表示形态, 对中间表示形态的掩体进行插桩处理, 加入若干功能函数调用; 最后, 将内存化处理后的目标软件和插桩处理后的掩体代码进行融合编译, 构建生成多样化保护处理后的目标软件, 该软件在静态特征上, 与目标软件完全不同, 难以通过静态逆向手段来获知目标软件的实现细节和关键代码; 在动态行为上, 保护后的软件兼具掩体行为, 能完全保持目标软件行为。

与传统代码保护方法相比, 本文提出的基于融合编译的软件多样化保护方法具有明显优势, 主要体现在: 与代码混淆方法相比, 该方法可以针对无源码的二进制文件, 对目标软件的所有指令和数据进行加密压缩处理, 达到完全混淆的效果; 与代码加壳相比, 该方法没有固定的加壳特

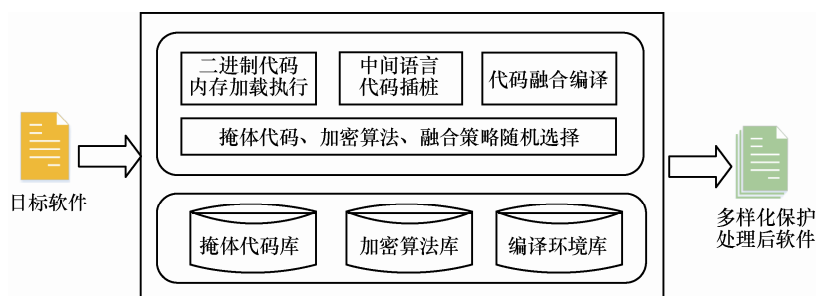


图1 软件多样化保护框架
Figure 1 Diversity software protection framework

征，而是利用多样化的掩体来加载和执行目标软件，难以通过脱壳方式来还原保护后的目标软件，在很大程度上增加了代码被逆向、被破解的难度。

4 基于融合编译的代码保护技术

在软件多样化保护框架设计过程中，涉及的关键技术主要有 PE 文件内存加载执行技术、中间表示代码插桩技术、掩体和目标代码深度融合编译技术。其中，PE 文件内存加载执行技术用来将目标软件组装成可在内存中执行的状态，为目标软件的全指令和全数据加密处理提供基础；中间表示代码插桩技术用来解决掩体代码融合点的选择问题，为融合编译提供基础；掩体和目标代码深度融合编译技术将掩体代码和目标软件代码，在中间表示层面进行深度融合，采用混合编译方法，生成用掩体程序保护后的目标软件。下面分别对各技术的核心思想进行介绍。

4.1 软件内存加载执行技术

软件内存加载执行技术解决的是目标软件在内存中直接执行的问题，使目标软件可被全文件加密保护，在执行时无须释放出 PE 文件而失去保护意义。

正常情况下，要执行一个 PE 文件，一般通

过双击或命令行，以创建进程的形式来完成，这些工作均依赖于 Windows PE 加载器^[20]。而内存执行技术是模拟加载器的功能，来实现 PE 程序的加载执行。简而言之，内存执行技术是在内存中，以无进程形式直接执行一个 PE 文件^[21]。PE 文件内存加载、执行是一个非常复杂的过程，涉及文件内存映射、导入表修复、重定位修复、TLS 段修复、PEB 结构修复等技术。本文在对 PE 文件加载运行机制进行深入分析的基础上，提出了一种通用的软件内存加载执行技术，有效解决了内存执行过程中出现的导入表缺失、PEB 结构错误、TLS 表丢失、地址随机化等问题，技术框架如图 2 所示。

首先，将目标文件加载到加载器所在内存空间中，完成内存区域扩展，对加载到内存中的 PE 文件进行重定位和导入表修复处理，使 PE 文件在内存中的虚拟地址能够被正确访问；接着，在加载 PE 文件的内存区域中进行 PEB 结构修复和 TLS 表修复，使目标代码在执行过程中能够获取 PEB 结构中的正确信息，保证程序的正常运行。

4.2 基于 LLVM 中间表示的代码插桩技术

本文选择底层虚拟机（LLVM, low level virtual machine）作为代码插桩平台，LLVM 是一种

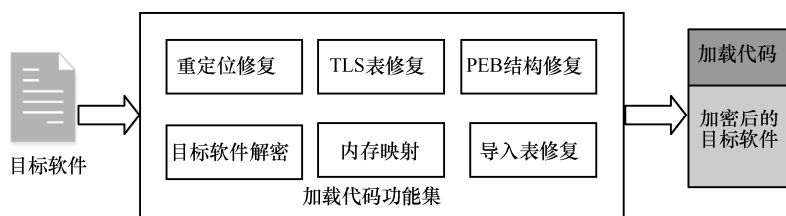


图2 软件内存执行技术框架
Figure 2 Software memory execution technology framework

成熟的开源编译框架, 提供了丰富、方便的中间表示编程接口。LLVM 最初是伊利诺伊大学的一个研究项目。它是一个模块化和可重用的编译器和工具技术的集合, 其核心库提供了与编译器相关的支持, 可以作为多种语言编译器的后台来使用, 能够进行程序语言的编译期优化、链接优化、在线编译优化、代码生成优化。

LLVM 中间表示是一种采用静态单赋值 (SSA, static single assignment) 形式的中间表达 (Intermediate Representation), 包含一套汇编语言类似的指令集, 类似于 RISC (reduced instruction set computer) 的三地址指令集, 具有简单的控制指令和带类型的访问内存指令, 独立于前端高级语言, 同时独立于目标平台架构。它既能保留源程序的信息 (用于程序分析和变换), 又足够简单便于程序优化。LLVM 为开发者提供了很多代码分析、编译优化相关的开发库, 为中间表示代码插桩提供了丰富、规范的接口支撑。

本文以开源编译框架 LLVM 为基础支撑, 将掩体代码编译成中间表示形态, 利用 LLVM 提供的中间表示编程接口, 对掩体代码进行插桩处理, 在掩体执行的关键路径中, 插入目标软件加载运行所需要的功能代码。在插桩过程中, 插桩点的选取直接影响融合后代码的正确性、融合过程的隐蔽性, 本文在选取插桩点时, 遵循路径必达、深度优先、随机选取等策略。插桩代码负责加载并执行加密后的目标软件, 技术细节如图 3 所示。

利用 LLVM 编译器, 将掩体代码编译成中间表示形态, 在中间表示层面记录程序的动态执行

轨迹, 每个轨迹点的内容包括函数名、基本块名、指令名、操作数序号、操作数的值等要素, 这些点构成了掩体融合点数据库, 确保在融合点处插入的功能代码都能被执行。插桩代码的功能主要包括解密、加载、内存执行、上下文衔接等, 负责实现对被保护目标软件的完整、正确执行。面向 LLVM 中间表示的代码插桩算法描述如下。

procedure IR_Instrument (IRFile): 面向 LLVM 中间表示的代码插桩算法

输入 待插桩的中间表示文件 IRFile, 插入的融合功能代码 FuseCode

输出 插桩后的中间表示文件 NewIRFile

begin

1) Module = ParseFile(IRFile); //列出文件中的所有函数

2) FunsArray = ListFuns(Module); //遍历所有函数

3) foreach curFun in FunsArray //遍历所有基本块

4) foreach curBB in BBsArray //遍历所有指令

5) foreach curIns in InsArray //是否包含值确定操作数

6) if HasValueDefinedOperand(curIns)
//插入融合点记录代码

7) NewModule=InsertRecord Code (Module);

8) end if

9) end foreach

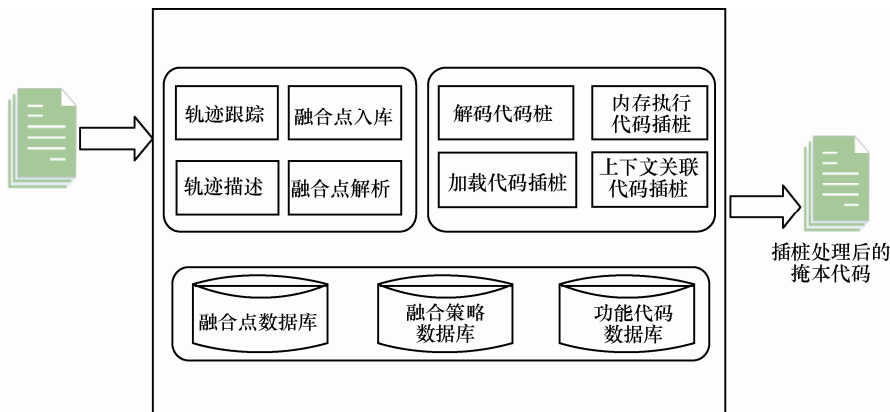


图 3 面向中间表示的代码插桩技术
Figure 3 Instrumentation technology based on LLVM IR

```

10)   end foreach
11) end foreach //编译插桩之后的中间表示文件
12) IREXE = CompileFile(NewModule); //
执行插桩后的 exe 文件，获取所有的融合点
13) PointArray = RunExe(IREXE);
//随机获取融合点，便于插入融合代码
14) SelectedPoints = GetRandPoints (PointAr-
ray); //在掩护代码融合点处插入融合代码
15) InsertFuseCode(IRFile, FuseCode, Selec-
tedPoints);
end

```

4.3 掩体和目标代码融合编译技术

在完成掩体代码插桩工作后，需要将掩体功能和被保护软件功能有机融合在一个程序中，该程序利用随机选择的掩体，从静态特征、动态行为等方面对目标软件进行全方位保护。本文提出了一种掩体和目标代码深度融合编译技术来完成上述目标，技术细节如图4所示。

插桩处理后的掩体代码被单独编译成目标对象 obj 文件，其中在融合点处嵌入了对目标软件解密、加载、执行等核心功能函数的调用，但这些函数的具体实现并不在掩体代码中，而是在单独的功能代码集合中，功能代码集合和加密后的目标软件统一编译成目标对象 obj 文件。最后，利用 LLVM 提供的交叉链接接口，将上述两部分代码编译生成一个最终文件，实现掩体代码和目标软件的充分融合，达到目标软件多样化保护目的。

在融合时，掩体能提供的融合点数量非常庞

大，为了使插桩过程尽量随机化，本文构建了深度、数量、分块算法、解密算法等均可随机配置的融合策略，即使对同一掩体，其插桩后生成的代码也有较大差异，利用掩体代码的多样性、融合策略的随机性可实现目标软件的多样化保护效果。

5 实验评估

为了验证基于融合编译的程序多样化保护方法的整体性能和实际效果，本文借鉴了软件保护效果的主流评估方法^[22]，从方法的时间、空间开销，保护前后代码相似性分析，与加壳类代码保护方法的比较等方面开展了实验分析。

5.1 测试集

本文选取 10 个常用系统软件、应用软件组成测试用例集，各用例详细说明如表 1 所示。测试主机环境为 32 GB 内存、Intel I7 处理器、Win 10 X64 系统。

5.2 资源开销分析

本文从两个方面来分析基于融合编译的程序多样化保护方法的资源开销情况：一是保护方法对目标软件进行保护处理时，所需要的时间开销，反映的是保护方法的运行效率；二是目标软件被保护处理前后的文件大小和执行时间对比，反映的是保护方法对目标软件运行效率的影响。

图 5 展示了在同一掩体和相同配置下，利用本文方法分别对 10 个测试用例进行保护处理的时间开销情况，从实测数据可以看出，保护方法所需时间与样本大小强相关，一般来说，目标软件越大，所需处理时间越长。总体来说，保护方

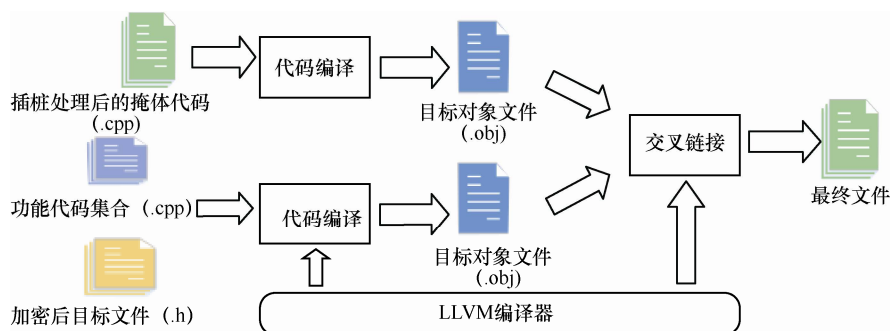


图4 掩体和目标软件融合编译过程
Figure 4 Compilation process of the fusion of bunker and target software

法时间开销较小。

在相同的掩体和配置参数下,利用本文提出的多样化保护方法对所有测试用例进行保护处理,生成新的软件。图 6 展示的是多样化处理前后,测试用例的执行时间对比情况,可以看出,保护后的目标软件执行时间较保护前有所增加,时间净增长值较为平稳,这主要是因为保护后的代码不仅要具备原始目标软件的功能,还要具备掩体程序的功能。同一个掩体,执行时间是相对固定的,但对于各测试用例,即使利用的是同一掩体进行保护,时间净增长值也各有差异,这主要是由掩体融合点和目标软件加密压缩算法的随机性因素引起的,这也充分体现出软件保护的多样化效果。

图 7 展示的是各测试用例保护前后文件大小变化情况,从图中可以看出,保护后文件大小较保护前有明显增长,增长部分主要是掩体代码,并没有引入其他过多额外代码,这与本文提出的软件保护方法的工作原理相吻合。

5.3 保护效果分析

为了说明所提出的软件保护方法的多样化特性,本文从两个角度开展实验分析:一是比较软件保护处理前后的代码相似性,说明软件保护技术本身的多样化特性;二是利用不同掩体对相同目标软件进行保护处理,比较不同掩体保护处理后的软件相似性,说明掩体对软件保护的多样化效果。一般来说,软件处理前后文件之间的相似性越低,说明保护方法的多样化效果越好。

表 1 测试用例说明
Table 1 Test case description

序号	用例名	文件大小	用例说明
1	systeminfo.exe	99 kB	Win10 系统自带程序,显示本地或远程机器(包括服务包级别)的操作系统配置的信息
2	net.exe	56 kB	Win10 系统自带程序,可用于管理账户、配置服务等
3	help.exe	12 kB	Win10 系统自带程序,用户显示 Windows 下的各种帮助信息
4	telnet.exe	129 kB	Win10 系统自带程序,用于连接外部特定主机和端口
5	taskkill.exe	24 kB	Win10 系统自带程序,用户结束指定的进程
6	where.exe	40 kB	Win10 系统自带程序,显示符合搜索模式的文件位置。在默认情况下,搜索是在当前目录和 PATH 环境变量指定的路径中执行的
7	rar.exe	584 kB	Windows 系统下常用压缩软件 WinRAR 5.7 的命令行文件
8	7z.exe	286 kB	Windows 系统下常用压缩软件 7z 的命令行文件
9	server.exe	79 kB	Windows 系统下一款常用开源远控软件
10	tcpsscanner.exe	56 kB	Windows 系统下一款基于 TCP 的端口扫描软件

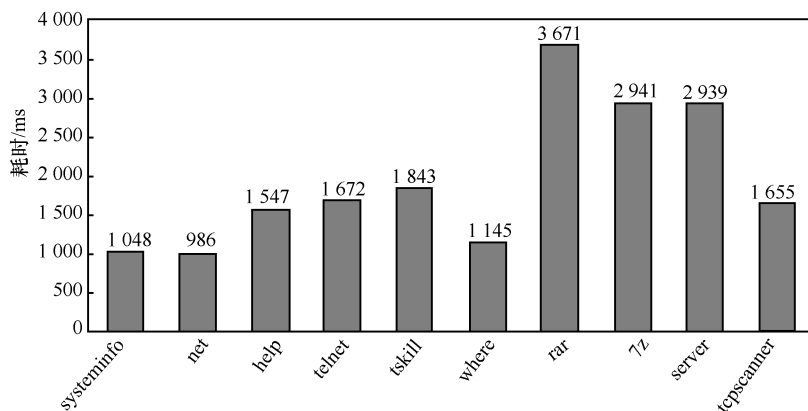


图 5 软件多样化保护处理时间开销
Figure 5 Time overhead of diversity software protection

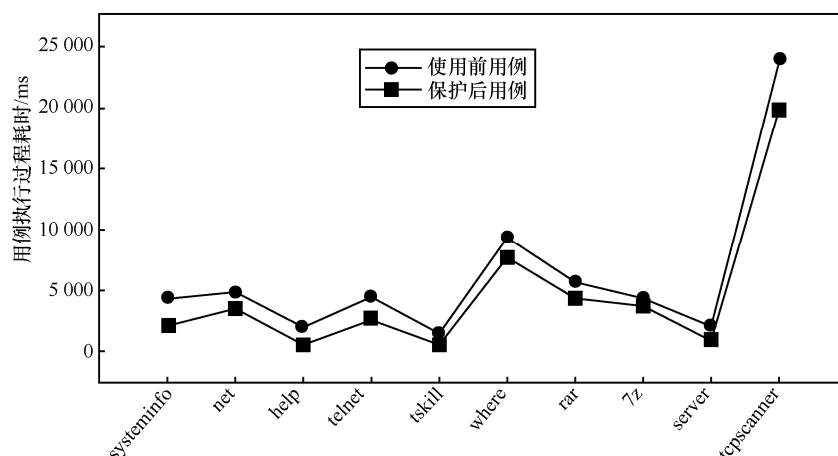


图6 软件多样化保护前后执行时间对比

Figure 6 Comparison of execution time before and after software protection

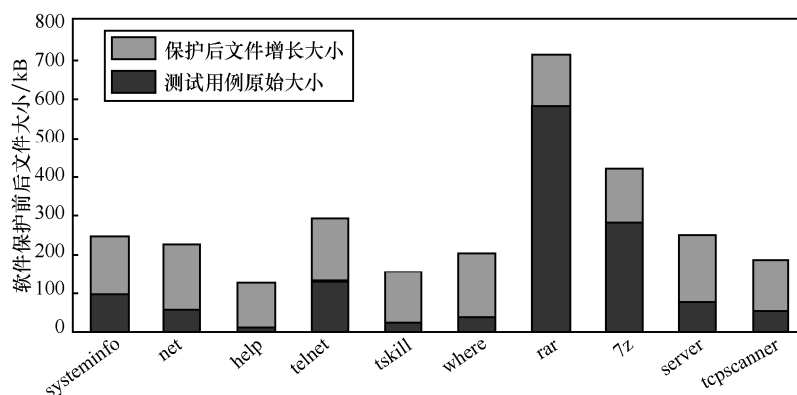


图7 软件多样化保护前后文件大小对比

Figure 7 Comparison of file size before and after software protection

二进制代码相似性比较技术常用的有四大类，分别是基于二进制字节比较、基于二进制文件反汇编后的文本比较、基于指令相似性的图形化比较和基于结构化签名的比较方法，每类方法各有特色。本文选用基于汇编指令的比对方法，对二进制文件进行反汇编处理，以汇编指令作为分析对象，分别提取指令操作码和操作数，进而对其进行相似性比对^[23]。本文选用常用二进制文件比对软件 BinDiff^[24]，从基本块和函数两个维度来计算软件保护前后文件之间的相似性。

在进行基本块和函数级文件相似性比较时，不考虑系统库或第三方库代码，只是比较软件自实现代码的相似性。表2是各测试用例在用本文方法进行保护前后的文件相似性统计情况，从中可以看出，无论从基本块级还是函数级，保护前后的代码相似性普遍很低，这说明保护前后软件

不具有同源性，体现了本文方法的多样性和抗同源性特点。

表2 各用例软件保护前后的文件相似性
Table 2 File similarity before and after protection of test case

用例	基本块相似性	函数相似性
systeminfo	5.3%	3.5%
net	2.4%	1.6%
help	3.1%	2.3%
telnet	5.2%	4.3%
tskill	2.6%	2.1%
where	3.9%	3.3%
rar	3.5%	2.8%
7z	4.3%	3.1%
server	6.2%	5.3%
tcpscanner	5.8%	4.3%

表 3 是利用不同掩体,对 3 个测试用例进行保护处理前后的文件相似性统计情况,可以看出,即使针对同一测试用例进行保护,由于选用的掩体不同,保护处理后得到的文件相似性很低。该统计结果与本文方法技术路线一致,在利用掩体代码对目标软件进行保护处理时,受保护的目标软件被随机加密处理,通过自实现的加载器来运行,而加载器设计成 shellcode 形态,也可被加密处理。上述过程不会引入额外的公共代码,所以利用不同掩体保护之后的目标软件呈现出很强的多样化特性。

表 3 利用不同掩体对软件进行保护前后的文件相似性
Table 3 File similarity before and after software protection with different bunkers

用例	比较维度	掩体 A 掩体 B	掩体 C 掩体 D	掩体 E 掩体 F
tskill	基本块	4.3%	5.4%	4.8%
	函数	3.2%	4.6%	4.2%
7z	基本块	4.7%	5.3%	3.9%
	函数	3.1%	4.8%	3.7%
server	基本块	3.6%	3.9%	4.4%
	函数	3.3%	3.1%	3.7%

5.4 对比实验分析

从技术思路和实现方法来看,本文方法与传统加壳保护方法存在较大差异。本文选取了 3 款常用加壳软件对测试用例进行加壳保护,从保护后的软件是否会被检测为恶意软件、是否能被

还原、是否能被检测出壳特征 3 个维度,来对比本文方法与传统加壳方法在软件保护效果方面的优劣。

本实验选取的 3 款常用加壳软件及其版本分别为 UPX3.4、ASPack2.12 和 ASProtect2.77。其中,UPX 是一款先进的可执行程序文件压缩器,压缩过的可执行文件体积缩小 50%~70%。通过 UPX 压缩过的程序和压缩之前程序功能完全一致。ASPack 是一款针对 Win32 可执行文件的常用加壳软件,使用方便,操作快捷。ASProtect 是一款功能非常完善的加壳、加密保护软件,能够在对软件加壳的同时进行各种保护,如反调试跟踪、自校验及加密保护等。

表 4 描述了本文提出的多样化软件保护方法与常用加壳类软件保护方法的效果对比情况。其中,A 表示处理后代码是否能被检测出壳特征,用 ExeInfo 工具检测;B 表示处理后代码被检测为恶意代码的概率,通过把代码上传至 VirusTotal 网站,用检出比来表示被检测概率;C 表示处理后代码是否能被还原,即能否利用专用脱壳工具或动态调试方法还原出原始软件形态。Y 表示“是”;N 表示“否”。可以看出,加壳类软件保护方法一般具有很明显的壳特征,壳代码形态相对固定,逆向分析人员可利用专用脱壳机对其进行脱壳处理,还原出加壳前的代码形态。另外,加壳处理后的代码会表现出较强的恶意性,容易被杀毒软件识别为恶意程序。针对上述问题,本文

表 4 与加壳软件保护方法的对比结果
Table 4 Compared with common software packing tools

测试用例	UPX 加壳			ASPack 加壳			ASProtect 加壳			多样化保护		
	A	B	C	A	B	C	A	B	C	A	B	C
systeminfo	Y	65%	Y	Y	52%	Y	Y	36%	Y	N	0	N
net	Y	53%	Y	Y	46%	Y	Y	38%	Y	N	0	N
help	Y	68%	Y	Y	40%	Y	Y	30%	Y	N	0	N
telnet	Y	48%	Y	Y	56%	Y	Y	42%	Y	N	0	N
tskill	Y	42%	Y	Y	54%	Y	Y	46%	Y	N	0	N
where	Y	42%	Y	Y	38%	Y	Y	36%	Y	N	0	N
rar	Y	51%	Y	Y	42%	Y	Y	30%	Y	N	0	N
7z	Y	54%	Y	Y	46%	Y	Y	28%	Y	N	0	N
server	Y	80%	Y	Y	76%	Y	Y	74%	Y	N	8%	N
tcpsscanner	Y	76%	Y	Y	72%	Y	Y	70%	Y	N	4%	N

提出的软件多样化保护方法具有明显优势, 不会产生明显的代码特征, 壳检测工具失效, 保护方法的多样性使无法利用专用脱壳工具对其进行还原处理。更为重要的是, 本文方法不会引入额外的恶意行为或恶意特征, 对于正常目标软件, 保护后的代码不会被检测为恶意代码。

6 结束语

软件保护技术一直是信息技术领域研究的热点和难点问题, 对于提高软件安全性、保护软件知识产权具有至关重要的作用。随着软件静态反汇编、动态跟踪调试等逆向分析技术的快速发展, 混淆、加壳等传统软件保护方法逐渐暴露出壳特征明显、易被还原、混淆模式单一、适用面不广等问题, 难以对抗日趋先进的软件逆向手段。本文以 LLVM 开源编译框架为基础, 提出了一种基于融合编译的软件多样化保护方法, 方法的核心思路是把要保护的目标软件与另一个掩护软件进行深度融合处理, 将目标软件的特征和行为分散在掩护代码中。首先将目标软件进行随机化加密处理, 并整体作为数据, 与掩护代码在编译层面进行深度融合; 然后通过内存加载执行技术, 将加密后的目标软件进行解密, 分阶段在内存中以无进程的形式运行起来, 通过掩护代码的多样性、融合点和融合策略的随机性来实现目标软件的多样化保护效果。对于同一个目标软件, 可以生成大量特征各异、形态多样的保护后软件。本文选取了多款常用软件作为测试集, 从资源开销、保护效果、对比实验等多个角度对本文方法的性能和效果进行了实例测试。从测试结果来看, 与混淆、加壳等传统保护方法相比, 所提方法在抗静态分析、抗动态调试、抗同源性分析等方面表现出较大优势, 资源开销较小, 多样化保护效果明显。

从原理上来讲, 没有无法被还原的软件, 也没有无法被破解的软件保护方法, 软件保护是一种具有典型对抗特征的软件技术, 该技术的发展与软件破解、软件逆向等技术强相关、紧耦合, 其核心思想是在保证执行效率的情况下, 尽量加大软件被逆向、被破解的难度。软件保护技术研究是信息技术领域永恒的话题, 在本文方法中,

掩护代码的随机性和多样性对于保护效果起着非常重要的作用, 接下来将继续优化基于融合编译的软件多样化保护方法, 进一步研究掩护代码的智能化构造方法, 为软件多样化保护提供丰富的掩护库支撑。

参考文献:

- [1] 周国祥, 陆文海. 基于 BHO 技术的数字版权保护系统的研究与设计[J]. 计算研究与发展, 2010, 47(6): 316-320.
ZHOU G X, LU W H. Application of BHO-based PDF digital management system[J]. Journal of Computing Research and Development, 2010, 47(6): 316-320.
- [2] 刘芳, 金松根, 卢国强. 数字版权管理与数字图书馆建设[J]. 图书馆学刊, 2011(4): 105-108.
LIU F, JIN S G, LU G Q. Digital copyright management and digital library construction[J]. Journal of Library Science, 2011(4): 105-108.
- [3] 国务院. 计算机软件保护条例[J]. 新疆新闻出版, 2013(2): 82-84.
State Council. Regulations of protecting computer software[J]. Publishing House of Xinjiang News, 2013(2): 82-84.
- [4] 刘军. 基于 CUDA 的软件保护技术研究与实现[D]. 长沙: 湖南大学, 2011: 5-25.
LIU J. The Research and implementation of CUDA-assisted software protection technology[D]. Changsha: Hunan University, 2011: 5-25.
- [5] 高艳军. 数据安全管理系统加壳技术研究与实现[D]. 长沙: 国防科学技术大学, 2008: 15-30.
GAO Y J. Research and Implementation of packing technique for data security management System[D]. Changsha: National University of Defense Technology, 2008: 15-30.
- [6] 马珂. 基于虚拟机的内核模块行为分析技术研究[D]. 湘潭: 湘潭大学, 2014: 20-28.
MA K. Research on the technologies of kernel module behavior analysis based on VMM[D]. Xiangtan: Xiangtan University, 2014: 20-28.
- [7] 郝景超. Windows 下软件实名机制的设计与实现[D]. 郑州: 解放军信息工程大学, 2008: 24-36.
HAO J C. Design and realization of the software real name mechanism in windows system[D]. Zhengzhou: PLA Information Engineering University, 2008: 24-36.
- [8] 李勇. 基于 Windows 平台的目标代码混淆[D]. 成都: 电子科技大学, 2007: 37-45.
LI Y. Object code obfuscation based on Windows platform[D]. Chengdu: University of Electronic Science and Technology of China, 2007: 37-45.
- [9] 房鼎盛, 张恒, 汤战勇, 等. 一种抗语义攻击的虚拟化软件保护方法[J]. 四川大学学报(工程科学版), 2017, 49(1): 159-168.
FANG D Y, ZHANG H, TANG Z Y, et al. DAS-VMP: a virtual machine-based software protection method for defending against semantic attacks[J]. Journal of Sichuan University (Engineering Science Edition), 2017, 49(1): 159-168.

- [10] TANG Z, LI G, FANG D, et al. Code virtualized protection system with instruction set randomization[J]. Journal of Huazhong University of Science & Technology, 2016,44(3):28-33.
- [11] KUANG K, TANG Z, GONG X, et al. Exploiting dynamic scheduling for VM-based code obfuscation[C]//Proc of the Trust-com/Bigdata/Ispa. 2017. 489-496.
- [12] COLLBERG C, THOMBORSON C, LOW D. A taxonomy of obfuscating transformations[J]. Department of Computer Science the University of Auckland New Zealand, 1997.
- [13] COLLBERG C, THOMBORSON C, LOW D. Manufacturing cheap, resilient, and stealthy Opaque constructs[C]// Proc. 25th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages '98. 1998: 184-196.
- [14] BANESCU S, COLLBERG C, GANESH V, et al. Code obfuscation against symbolic execution attacks[C]//Proceedings of the 32nd Annual Conference on Computer Security Applications. 2016: 189-200.
- [15] WANG Z, JIA C F, LIU W J, et al. Branch obfuscation to combat symbolic execution[J]. Acta Electronica Sinica, 2015,43(5): 870-878.
- [16] 潘雁,祝跃飞,林伟.基于指令交换的代码混淆方法[J].软件学报,2019,30(6):1778-1792.
PAN Y, ZHU Y F, LIN W. Code obfuscation based on instructions swapping[J]. Journal of Software, 2019, 30(6): 1788-1792.
- [17] BALACHANDRAN V, KEONG N W, EMMANUEL S. Function level control flow obfuscation for software security[C]//Eighth International Conference on Complex, Intelligence and Software Intensive Systems. 2014:133-140.
- [18] 王志. 二进制代码路径混淆技术研究[D]. 天津: 南开大学. 2012.
WANG Z. Research on binary code path obfuscation[D]. Tianjin: Nankai University. 2012.
- [19] TAMBOLI T, AUSTIN T H, STAMP M. Metamorphic code generation from LLVM bytecode[J]. Journal of Computer Virology & Hacking Techniques, 2014, 10(3):177-187.
- [20] 章立春. 一种 PE 程序文件加载执行方法, CN 104331308 A[P]. 2015.
- ZHANG L C. A method for loading and executing PE program files, CN 104331308 A[P]. 2015.
- [21] ZHOU J. Method and apparatus for bypassing hook: CN 101414338 B[P]. 2012.
- [22] WANG H, FANG D, LI J, et al. The research and discussion on effectiveness evaluation of software protection[C]//Proc of the Int'l Conf on Computational Intelligence and Security. 2016. 628-632.
- [23] SCHULMAN A. Finding binary clones with opstrings and function digests[J]. Dr.Dobb's Journal, 2005, 30(9): 64-70.
- [24] ZYNAMICS. Sabre BinDiff[EB].

[作者简介]



熊小兵 (1985-), 男, 江西丰城人, 博士, 信息工程大学讲师, 主要研究方向为网络信息安全、软件逆向分析。



舒辉 (1974-), 男, 江苏盐城人, 博士, 信息工程大学教授、博士生导师, 主要研究方向为网络信息安全、嵌入式系统分析与信息安全。



康缙 (1972-), 女, 北京人, 信息工程大学教授, 主要研究方向为网络信息安全、加解密机制分析。