随机森林在程序分支混淆中的应用

陈 喆[1],贾春福[1,2,3],宗 楠[1],郑万通[1]
(1. 南开大学计算机与控制工程学院,天津 300350;2. 中国民航大学信息安全测评中心,天津 300300;
3. 河北省高可信信息系统重点实验室,河北保定 071002)

摘 要: 程序中的路径信息在程序执行过程中会被动态地泄露,基于路径敏感技术的逆向工程可自动地收集程序中的条件跳转指令,从而理解程序的内部逻辑.为了缓解路径信息泄露,提出了一种基于随机森林的路径分支混淆方法,将逆向分析路径分支信息的难度等价于抽取随机森林规则的难度.鉴于随机森林分类器可被视为一种黑盒,其内部规则难以被提取且分类过程与路径分支行为相似,因此经过特殊训练的随机森林可以在功能上替代路径分支.将该方法部署于 SPECint-2006 标准测试集中的 6 个程序进行实验,实验结果表明该混淆方法有效地保护了路径分支信息,引发的额外开销较低,具有实用性.

关键词: 代码混淆;逆向工程;随机森林
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2018)10-2458-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2018.10.020

Branch Obfuscation Using Random Forest

CHEN Zhe[1], JIA Chun-fu[1,2,3], ZONG Nan[1], ZHENG Wan-tong[1]
(1. *College of Computer and Control Engineering, Nankai University, Tianjin* 300350, *China*;
2. *Information Security Evaluation Center of Civil Aviation, Civil Aviation University of China, Tianjin* 300300, *China*;
3. *Key Lab on High Trusted Information System in Hebei Province, Baoding, Hebei* 071002, *China*)

Abstract: Reverse engineering can automatically collect the path information which has been leaked in program runtime, and then reveal the internal logics of programs. To mitigate path information leaking, this paper proposes a novel path obfuscator to combat state-of-art reverse engineering by using random forest. The difficulty of reversing the obfuscated branch is equivalent to extracting rules from random forests. Since random forests can be regarded as black-box and the categorizing process has high similarity with the behavior of path branch, so the specially trained random forest can realize the functionality of path branch. The proposed obfuscator had been deployed on six programs from SPECint-2006 benchmarks. The experimental results show that our method has significantly increased the computational cost of reverse engineering, and the introduced overhead is acceptable.

Key words: code obfuscation; reverse engineering; random forest

1 引言

代码混淆技术是一种保留语义的程序变换,其目的是增加逆向工程的难度,使程序逻辑变得"难以被理解",从而保护软件的知识产权. Collberg 等人[1]对代码混淆技术做了形式化的定义:经混淆变换后的程序 O(P)应与 P 有相同的语义,在相同输入的情况下,P 和 O(P)会计算出相同的输出. Barak 等人[2]通过计算复杂性和密码学原理,证明了理想的混淆并不存在. Beaucamp 等人[3]从实用角度证明了代码混淆技术仍是保护程序的重要手段.在软件保护领域,代码混淆一直是研究的热点.

现今,软件知识产权的主要威胁来自于非受控环境下的恶意主机,即文献[4]中所提出的 MATE(Man-At-The-End)攻击. MATE 攻击以逆向工程为基础,通过逆向分析软件的可执行文件,对软件的知识产权造成多种形式的危害,如代码盗用、恶意篡改、盗版等.在发

收稿日期:2016-12-10;修回日期:2017-11-28;责任编辑:孙瑶
基金项目:国家重点基础研究发展计划(No. 2013CB834204);国家自然科学基金(No. 61772291);天津市自然科学基金(No. 17JCZDJC30500);中国民航大学信息安全测评中心开放课题(No. CAAC-ISECCA-201702)

布的可执行文件中,程序的路径信息在二进制代码上存在严重的泄露问题. 如图 1 所示,图中的汇编代码泄露了路径分支的全部信息:分支点、分支条件和目标地址. 逆向分析工具可以通过收集这些信息轻易地揭示代码块间的依赖关系,从而构建控制流图,提取控制流信息.

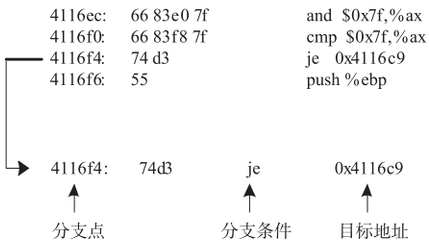

图1　路径信息泄露

保护路径分支信息的有效手段是代码混淆,最初的分支混淆方法是用来迷惑静态反汇编[5,7],但是这些经典的方法对动态分析和符号执行攻击的抵抗力较弱. 为了对抗这些新型攻击技术,本文提出了利用经过特殊训练的随机森林混淆路径分支的方法. 相比于其他常用分类器,如人工神经网络和支持向量机,随机森林具有更广的适用范围和更高的训练执行效率. 路径分支的功能类似于机器学习中的分类器,将路径分支的输入作为样本,输出作为标签,训练出的随机森林可以在功能上替代路径分支. 根据已有的研究成果,随机森林可被视为一种黑盒,内部的非线性规则难以被抽取,这是本文所提出的混淆方法的安全基础. 利用无条件跳转替代条件跳转,可以在静态分析中模糊程序的控制流,从而隐藏路径分支结构,增加恢复原有控制流的难度. 通过密码学方法混淆分支输入集合的偏序关系,使得加密后的分支输入具有一定的密码学特性,用来抵抗暴力破解攻击和二分查找攻击. 本文将提出的混淆方法部署在 SPECint-2006 标准程序测试集上,对路径分支的混淆强度和执行效率进行了测试,选用了最新的符号执行工具 S2E[8] 作为约束求解器来分析被混淆的路径分支. 实验评估显示本文的方法具有较强的攻击容忍性,且引入的额外开销较低,具有较好的实用性.

2　相关工作

2.1　符号执行及其应用

符号执行[8~10]是一种软件测试方法,它利用符号替代真实数值分析程序,求解触发特定路径的输入. 经过三十多年的发展,当今的符号执行技术可以自动地生成测试输入,尽可能多地覆盖程序所有的执行路径[12,13]. 利用符号执行的这些优势,Brumley[14] 开发了一款名为 Mine-Sweeper 的工具用以检测恶意代码的触发条件;Sharif [15] 开发了用以自动分析恶意代码的模拟器;Wang[16] 结合了污点分析技术和符号执行技术来检测和移除恶意代码壳中的环境敏感代码;Zeng[17] 利用动态符号执行实现了被混淆代码的代码复用.

虽然这些逆向分析技术功能强大,但是仍存在一定局限性[18]:当路径约束条件难以被规约为确定的布尔表达式时,约束求解器就无法对其进行求解.

2.2　路径分支混淆

为了对抗这些先进的分析技术,研究人员提出了很多有效的方法. Sharif 等人[19] 选用了哈希函数加密路径分支,攻击者分析被混淆分支的难度等价于分析哈希函数的难度. 但是哈希函数不具备保序性,只能良好地混淆分支条件为等于的路径分支,对于条件为不等关系(如“<”或“>”)的混淆相对繁琐,会造成极大的空间和时间开销. Wang 等人[20] 对其 Sharif 的混淆方法进行了优化改进,相对地扩展了利用哈希函数混淆条件分支的适用范围.

Wang 等人[21] 摒弃了密码学算法,而利用未解数学难题克拉兹(Collazt)猜想混淆恶意代码的触发条件. 任何正整数输入通过“$3x+1$”变换后必定收敛为 1,约束求解器无法求出正确的输入. 但是在路径分支条件较为宽泛的情况下,约束求解器求解成功的概率会提高.

文献[22,23]均将机器学习中的分类器用于路径分支混淆,利用支持向量机和人工神经网络的内部规则难以被理解这一特性,混淆程序的路径分支. 但是这两种方法在实用性和安全性上均存在一定的问题:①受制于这两种分类器的先天特性,对于混淆离散语义的路径分支条件,诸如 $x>5$ && $y<5$ 等,这两种分类器均无法良好的处理,在一定程度上限制了两种混淆方式的应用场景;②在安全性上,这两种方法保留了输入集合内的偏序关系,攻击者只需进行二分查找攻击和暴力破解攻击便可以在有限计算次数内找到路径分支条件.

2.3　随机森林

随机森林[24] 是一种基于“Bagging”思想的非线性分类算法,由 Breiman 在 2001 年提出,现今已经成为应用非常广泛的机器学习算法,具有强大的学习和分类能力[25]. 随机森林是一个由许多棵决策树组合而成的分类器,每棵决策树由有放回随机抽取的随机向量生成,互相之间是独立的,分类结果由每棵树输出的类别的众数决定.

对于分类器内部规则的理解,近二十年一直是机器学习领域的热点研究方向[26,27]. 随机森林的内部规则是一种高维规则,难以被提取,因此可以将随机森林视为一种黑盒分类器[24,28~30]. 随机森林规则难以提取的根本原因是高维规则难以规约为低维规则,即粗糙

集中的属性约减难题[31,32],该问题已经被证明是一个NP问题,现今的研究成果无法准确地提取出随机森林的内部规则.随机森林的这一特性在机器学习领域是一个困扰,但是在软件保护领域该特性正是代码混淆所需要的.另外,随机森林具有训练速度快,对高维样本输入适应性良好的特点,这使得构建混淆模块的过程具有较高的效率.

3 使用随机森林混淆路径分支

3.1 系统框架

由图2所示,从一个独特的视角去看程序的路径分支,可以将其视为机器学习中的分类问题.对于一个路径分支,可以将程序的所有输入视为待分类的样本,分支条件视为分类器规则,产生的两条路径视为分类器的输出.基于分类器的工作过程,分类器可以在功能上替代程序的路径分支.

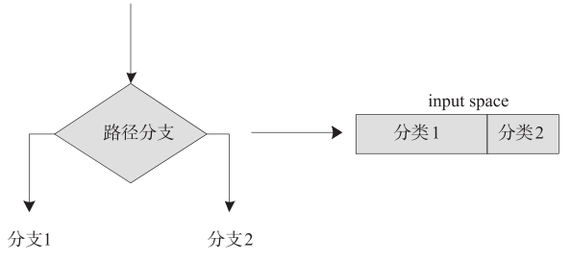

图2 路径分支与分类问题相似

本文选取了随机森林作为分支混淆算法,以期望达到适用更多类型的路径分支和更高的执行效率这两个目的.图3给出了使用随机森林混淆程序路径分支的框架.混淆后路径条件由随机森林的内部规则替代.在程序执行时,随机森林接收到路径分支的输入,通过训练好的规则输出相应的目标地址,触发对应的执行路径.

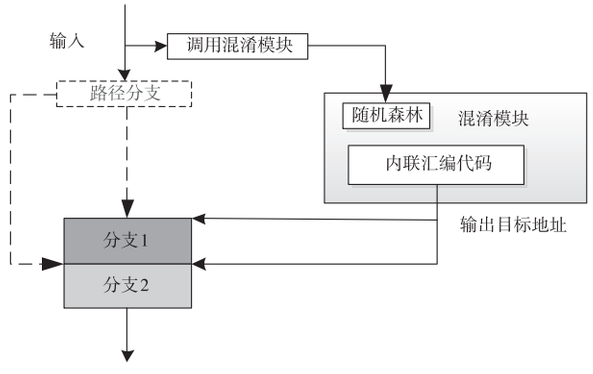

图3 应用随机森林混淆路径分支的框架

3.2 误差控制

在机器学习中,误差是考察分类器的重要指标,它表示分类器对训练集之外数据的分类能力,但是本文利用随机森林代替程序的路径分支,因此这种误差不能出现在程序运行过程中.为了保证混淆后的软件始终要运行在正确的执行路径中,Collberg[1]定义了语义保留这一要求:在给予相同的输入的情况下,被混淆的程序要和原程序计算出相同的输出,程序出错也视为一种输出.为了避免这种误差,本文根据Collberg提出的语义保留要求,结合分类器的特性,定义了弱语义保留这一新的要求.

定义(弱语义保留) 给定的两个分支 bF_{β} 和 bF'_{β} 其共同的输入集是有限的,若存在一个足够小的数值 ε 使得 $bF'_{(\beta\pm\varepsilon)}=bF_{\beta}$,那么就称分支 bF'_{β} 对分支 bF_{β} 是弱语义保留的.

定义弱语义保留是为了保证程序运行的正确性.该定义说明在可接受的偏差 ε 之内.近似的规则可以表现出相同的功能.偏差 ε 的大小应足够区分相邻的输入.例如,路径分支条件 $x\leqslant 0$ 和 $x<0.1$ 为两个截然不同的条件.但是如果分支输入为整数,即相邻输入为整数0和1时,这两个条件的功能是相同的,可以相互替代.这也就是说,在分支条件边界清晰的情况下,$x<0.1$ 这个分支条件所表达的语义与 $x\leqslant 0$ 相同,为Collberg所定义的语义保留的一种特殊形式.

在机器学习的场景下,分类器是用来使用已有的知识去预测训练集之外的数据,这样就会产生一定的误差.不同的是,在本文对随机森林的使用中,将路径分支所有可能的输入作为知识记忆在随机森林模型中,程序运行时,该分支的所有输入都不会在训练集之外,所训练的随机森林对训练集是过拟合的,这样就保证了被混淆后的程序会正确运行.可以确信,随机森林所包含的规则与原路径分支条件的差别一定在 ε 之内,符合本文定义的弱语义保留.

3.3 混淆偏序关系

文献[22,23]的工作保留了分支输入集合的偏序关系.在这种情况下,对混淆分支进行二分查找攻击,便可以在有限计算次数内找到分支条件的分界点.从兼顾安全性和实用性的考虑出发,本文选取了非对称Feistel网络混淆程序分支的输入集合.Feistel网络是一种在对称密码学中常见的单元结构,它以轮为单位,通过对输入数据进行分割迭代操作,使得密文的分布更加接近于均匀分布.通过Feistel网络,输入数据会被充分的混乱和扩散,增加攻击被混淆分支的难度.

4 基于随机森林的路径分支混淆的实现

本文在源代码的if-else路径分支条件中实现混淆,使用支持内联汇编的编译器将混淆后的代码编译为可执行程序.图4所示,具体的实现分为4步:提取分支信息,训练随机森林,重构源代码和编译.

第一步,先将原程序代码编译成可执行程序,利用静态分析工具定位待混淆的路径分支,提取分支条件和分支目标地址.

第二步,将分支输入集合通过 Feistel 网络进行加密置换,构建训练集,训练随机森林.

第三步,将随机森林模块写入源代码中替代路径分支,插入内联汇编代码恢复路径选择功能.

第四步,关闭某些优化条件对源代码进行编译.

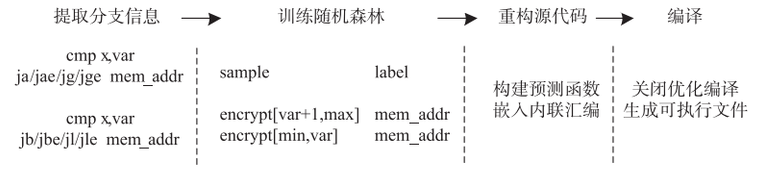

图4　混淆实现步骤

4.1　路径分支信息抽取

在可执行文件中,路径分支结构表现为条件跳转指令,如 jnz,ja,je 等. if-else 路径分支结构的 C/C++ 代码如下:

```
if(x>5)
  function A
else
  function B
```

编译为可执行程序后,反汇编代码如下:

```
text:00411A65 cmp[ebi+i],5
text:00411A69 jle short loc_411A72
text:00411A6B call j_function_A@@YAZZ
text:00411A70 jmp short loc_411A77
text:00411A72 call j_function_B@@YAZZ
```

通过反汇编代码收集有关该路径分支的所有信息:分支条件 $x>5$,两个分支目标地址 00411A6B 和 00411A72. 这些暴露的路径信息需要被当作知识隐藏在随机森林的内部规则中.

为了不失普遍性,本文将其他路径分支条件变视为 if-else 结构的特殊形式,将这些条件变换为相同语义的 if-else 结构,并按照本节的方法提取路径信息. 下面给出常见的复杂路径分支条件的处理方法.

(1)多输入变量的路径分支条件

例如 $x>y$,此类分支条件具有多个输入变量,分支判断结果随输入变量的变化而变化,难以辨别产生路径分支的边界条件. 在处理此类分支条件时,将条件改写为这些变量的差值与 0 比较,即 $x-y>0$. 这种处理方式将 $x-y$ 视为 1 个输入变量,处理后的分支条件边界清晰,易于构建训练集.

(2)for,while 的循环结构

此类循环结构,在判别是否跳出循环时产生路径分支. 可以将循环结构改为如下形式.

对于 for 循环:

```
for(变量;;变量操作)
  { if(循环跳出条件)
    循环体
  else
    跳出循环
  }
```

对于 while 循环:

```
while()
{
  if(循环跳出条件)
    循环体
  else
    跳出循环
}
```

处理后,循环结构变成死循环,跳出循环的条件被写为 if-else 结构,这样编译后的二进制代码中,分支结构更为清晰,利于提取路径信息.

4.2　随机森林训练

Librf [33] 由加州大学-圣塔芭芭拉分校的科研人员进行开发,经过多个版本优化,是拥有较高执行效率的开源代码库,为了降低混淆对程序执行效率带来的影响,本文选取该库中代码实现随机森林模块. 本文对 Librf 样例中的 rf-predict 文件中的代码进行了二次开发,删除了 rf. test_confusion、rf. testing_accuracy 等与预测行为无关的函数,修改了预测函数 rf. predict_prob 的输入参数,使其可以对程序变量进行预测. 假设被混淆的路径分支的输入集合为 $[a,5]\cup[6,b]$,使用 Feistel 网络加密后,构建格式为“. csv”的训练数据. 相应地,将两条分支地址 00411A6B 和 00411A72 作为对应的 Label 标签文件写入 data 文件夹中. 在训练时用训练集作为测试集测试训练结果,以确保得到 100% 正确率的分类结果. 训练集设置如下:

```
label:00411A6B Feistel[a,5]
label:00411A72 Feistel[6,b]
```

使用 Librf 中的 rf-train 程序对训练集进行训练,训练后提取得到的模型 model 文件中的参数待用.

4.3 源代码重建

与 Librf 的用法不同,本文将每棵决策树的参数通过对应的数组进行存储,无需在程序的命令行参数中输入单独的模型文件. 修改 read.in 函数的输入参数为数组类型,令其读取决策树数组中的数据作为随机森林的参数. 本文将随机森林的预测模块编写为一个 C/C++函数:predict 插入在源代码中,该函数的输入为加密后的分支输入,返回值为32 位的整型数,即两条路径分支的地址. 在 x86 结构的 CPU 指令集中,函数的返回值通常由寄存器 eax 保存,本文在预测函数后插入无条件跳转指令 jmp eax 用来完成路径选择功能. 当 predict 函数完成分类输出目标地址时,其返回的数值被存储在寄存器 eax 中,指令 jmp eax 可以完成相应的跳转功能. 内联汇编中的空指令 nop 是为了保证在编译后代码块的地址不发生改变而插入的. 重建后的源代码如下:

```
int predict (Feistel(x));
_asm_ _volatile_
{
nop
nop
jmp eax
}
function A;
goto label;
function B;
lable:
```

4.4 编译

任何支持 C/C++的编译器均可用来编译混淆后的源代码. 重建后的源代码中,function B 在静态中是一行死代码,某些编译器的优化选项不会将该代码编译至可执行文件而造成错误,这需要在编译时关闭某些优化措施.

5 评估

本节将对随机森林路径混淆方法进行安全性评估和效能评估. 在安全性评估方面,本文考虑了常用的逆向分析方法,利用逆向分析工具分析被混淆的分支,通过分析结果评估混淆方法的安全性. 在效能评估方面,本文在相同条件下对比了随机森林,SVM 和神经网络混淆对程序效能的影响. 为了得到贴近实际的评估,本文选取了 SPECint-2006 标准测试集中的 6 个由 C 语言编写的程序进行评估实验,分别是 specrand、sjeng、mcf、lbm、bzip2 和 h264ref. 对每个测试程序,随机选取了不同类型的路径分支,将每个分支条件输入集合通过非对称 Feistel 网络[34]加密并分割为多维向量作为混淆后的输入. 随着随机森林内部决策树的规模增大,随机森林的预测准确性会逐渐提高,选取 100 作为内部决策树的数量是兼顾效率和预测准确率的选择[35,36],因此用来混淆的随机森林含有 100 棵决策树. 表 1 标明了被混淆的分支条件和其在源代码中的位置.

表 1 被混淆分支的位置

测试程序	分支位置	分支条件
specrand	line 38 in specrand.c	test > 0
sjeng	line 1604 in search.c	x ! = 0
mcf	line 89 in mcf.c	if(new_arcs < 0)
lbm	line 166 in lbm.c	if (z = =0 ‖ z = =SIZE_Z - 1)
bzip2	line 555 in compress.c	if (s→inUse[i])
h264ref	line 93 in slice.c	if(i = =0)

实验环境:处理器为 Intel Core2 Q9400 2.66GHz,内存为 3GB RAM,操作系统为 Ubuntu 14.04,虚拟环境为 Ubuntu 12.04.

5.1 安全性评估

本文主要目的是阻止攻击者利用逆向分析技术理解被混淆的路径分支逻辑,使攻击者分析路径分支的开销尽可能的大. 即使攻击者多次分析,也不能从分析结果中得到正确的分支条件. 本文考虑了三种攻击者常用的攻击方式:静态分析,符号执行与约束求解,暴力破解与二分查找攻击.

(1)对抗静态分析

攻击者利用静态反汇编工具去反编译可执行程序,试图理解代码块之间的联系,和相应的控制流转移信息. 对 specrand 程序分支进行静态分析时,得到的反汇编代码如图 5.

```
.text:0040112D      mov    eax, [ebp+var_8]
.text:00401130      call   feistel
.text:00401135      mov    [ebp+var_8], eax
.text:00401138      call   predict
.text:0040113D      jmp    eax
.text:0040113D      sub_4010F0    endp
.text:0040113D
.text:0040113F      ;---------------------------------------
.text:0040113F      mov    ecx, [ebp-8]
.text:00401142      mov    dword_403374, ecx
.text:00401148      jmp    short loc_401159
.text:0040114A      ;---------------------------------------
.text:0040114A      mov    edx, [ebp-8]
.text:0040114D      add    edx, 7FFFFFFFh
.text:00401153      mov    dword_403374, edx
.text:00401159
.text:00401159      loc_401159:            ; CODE XREF: .text:00401148↑j.
.text:00401159      fild   dword_403374
```

图5 混淆分支的反汇编代码

根据图 5 中的代码可知在静态分析中,攻击者可以获得的信息仅仅为预测函数和插入的内联汇编代码. 在程序未运行时,寄存器 eax 中所存储的数值通常为0 或者其他无意义的数值,攻击者只能发现在jmp

eax 后将要执行的指令与预测函数的返回值有关. 图 6 为混淆分支的函数调用图,由于插入的无条件跳转指令 jmp eax 割裂了代码块之间的联系,静态分析工具探索到预测函数 predict 以下的控制流和函数调用. 如果想得到完整的路径信息,攻击者只能去分析预测函数内的代码. 图 7 为预测函数 predict 的内部核心函数 librf::InstantSet流程图,由该图可以看出,所有与预测过程相关的函数调用都要最终归结于这个核心函数,即调取随机森林参数. 随机森林所具有的黑盒特性可以使攻击者难以提取随机森林规则,因此本文的方法对基于反汇编的静态分析有较强的抵抗能力.

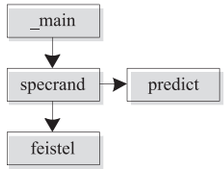

图6　混淆分支的控制流和函数调用

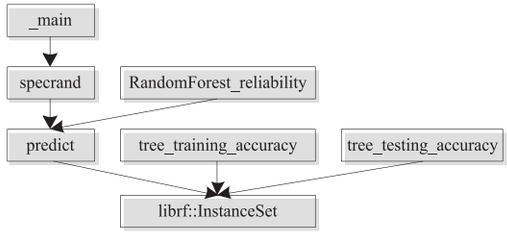

图7　predict内部核心函数librf::InstantSet的流程图

(2)对抗符号执行与约束求解

众所周知,符号执行和约束求解的局限性在于无法对非线性关系做出良好的求解结果. 随机森林是一种典型的非线性分类器,分类结果由投票最多的类决定. 分类结果和每棵决策树所输出的结果无线性关联,当输入变化时,得到的投票比也随之变化,分类过程难以被表达为确定的布尔表达式. 本文选用了先进的二进制分析平台 S2E 作为测试工具对被混淆的代码进行分析,将被混淆代码的输入进行标记,探索其执行路径,得到路径约束进行求解. 表 2 为对测试程序进行符号执行的结果. 实验结果表明,由于随机森林分类的机制,S2E 无法探索被混淆路径分支而导致停机或超出内存,约束求解器无法对混淆后的路径分支进行约束求解,本文提出的混淆方法对当前流行的自动分析工具具有较强的抗逆向分析能力,良好地抵抗了符号执行和约束求解这一新型的逆向分析技术. 另外,本文测试了不同规模随机森林对安全性的影响. 测试了内含 20,50,150,300,400,500 棵决策树的随机森林(忽略因随机森林规模较小所造成的误差,即无法过拟合训练集数据)进行符号执行测试,路径探测结果与表 2 中完全一致. 这印证了随机森林对抗符号执行与约束求解具有良好效果的核心原因是随机森林自身的黑盒特性,而非通过堆砌大量的决策树而导致程序路径看上去的“复杂”.

表 2　Spec-2006int 符号执行结果

测试程序	符号值地址	路径探索结果	求解结果
specrand	0x804863d	terminate	N/A
sjeng	0x805918a	terminate	N/A
mcf	0x804924c	terminate	N/A
lbm	N/A	out of memory	N/A
bzip2	0xbf91633c	terminate	N/A
h264ref	0x8b8379e	terminate	N/A

本文同时对比了文献[22,23]的方法在 S2E 平台的表现. 由于存在大量的浮点数计算,S2E 无法标记 SVM 中的路径分支变量. 由于 S2E 无法探索路径的原因与本文不同,因此无法定性地对比本文与文献[22]在对抗符号执行与约束求解的优劣性. 在对比文献[23]方法中时,S2E 可以探索到被标记变量所经过的路径,但是无法求解触发另一条路径所需的输入变量. 相对而言,随机森林阻止了符号执行工具探索标记变量所经过的路径,在对抗符号执行和约束求解的抵抗力上,本文的实验结果稍优于文献[23]的结果.

(3)对抗二分查找攻击和暴力破解攻击

由于本文提出的混淆方法的应用场景是在路径分支输入有限的情况下,所以应当考虑对这两种攻击的抵抗能力. 若某一路径分支的所有输入个数为 k,则二分查找攻击至多需要 $\log_2 k$ 次测试即可得出正确的数据分类边界. 例如,输入集合为[0,65536],分支条件为 5,测试输入为 10000. 二分查找的路径为(5000,2500,1250,625,312,156,78,39,17,8,4,6,5),只需 13 次测试便可找到正确输入. 本文选取了非对称 Feistel 网络混淆程序的输入集合,输入集合中的元素经过 Feistel 网络加密后,其原本的偏序关系被打乱,加密后的输入呈现随机性和扩散性. 攻击者在攻击被混淆分支时,二分查找攻击的测试次数会大幅增加,且效果较差,因此这种攻击方式难以进行.

在对抗暴力破解的能力上,本文对加密后的分支输入切分为若干个组块,相当于增加了随机森林模块的输入长度,一定程度上降低了暴力破解成功的概率. 穷举成功的概率为 $1/2^l$,其中 l 为加密后分支的输入长度,随着输入长度 l 的增加,破解成功的概率呈指数级下降.

本文选取了 specrand 程序中的分支来测试本文的方法对抗暴力破解攻击和二分查找攻击的抵抗力,并与其他两种混淆方法进行了对比. 在暴力破解攻击下,

多次运行测试程序,对输入进行随机不重复测试,直到测得满足分支条件的输入;在二分查找攻击下,随机生成初始输入. 表 3 给出了暴力破解攻击所需的测试次数、二分查找攻击所需的测试次数和二分查找的初始随机输入.

表 3 抵抗暴力破解攻击和二分查找攻击的能力

测试程序	暴力破解所需次数	二分查找所需次数	初始测试输入
specrand	25359	15	4875
随机森林混淆	186875	192084	74978
SVM 混淆	24571	11	730
神经网络混淆	26754	9	256

从表 3 数据可以看出,SVM 混淆和神经网络混淆的测试数据中,暴力破解攻击所需要的计算次数与原程序无较大差异,二分查找攻击次数与初始输入呈明显的对数关系,并未明显体现对这两种攻击的抵抗力. 本文工作中,暴力破解和二分查找攻击所需计算次数显著增加,二分查找的路径无法匹配加密后的输入,因此本文的方法对这两种攻击具有较强的抵抗力.

5.2 效能评估

本文采取了多次运行求平均值的方式计算被混淆程序的时间开销,同时统计了被混淆分支的执行次数. 对比算法中,分类器的训练方法均与原文献中的设置相同. 表 4 和表 5 分别列出了时间开销和空间开销数据. 由表 4 数据可见,时间开销与执行次数成正相关,specrand 中的被混淆分支执行次数为 48478 次,引入了较大的开销,其余测试程序的额外开销较小,平均每次执行所需开销约为 0.005s. 由于随机森林对高维数据处理具有优势,且随机森林内部计算由简单的分支判断构成,本文提出的混淆方法在时间开销上较优于另两种混淆方法. 本文加密了输入集合,文献[22,23]所提出的边界关键变量训练法无法应用于本文的场景. 使用加密后的输入集合作为训练集,导致这两种的时间开销和空间较原文献中的数据有较大增长. 表 5 中,由于随机森林的参数要保存在程序中,引入的额外空间开销较对比算法稍大. 但是对于当今硬盘容量(数百 GB 至几 TB),这些开销是可以忽略不计的. 本文提出的混淆方法在效能上是可以接受的.

表 4 时间开销对比

测试程序	原始开销	随机森林混淆	SVM 混淆	神经网络混淆	分支执行次数
specrand	0.801280s	11.150352s	16.732395s	20.638s	48478
sjeng	8.410403s	8.794101s	9.814571s	13.436s	2
mcf	3.978625s	4.186864s	4.926473s	4.8835s	1
lbm	6.178057s	6.154750s	6.196531s	6.35427s	1
bzip2	0.090153s	0.157843s	0.469917s	1.1455s	256
h264ref	51.113300s	62.125628s	70.518473s	69.938s	25

表 5 空间开销对比

测试程序	原始开销	随机森林混淆	SVM 混淆	神经网络混淆
specrand	16KB	112KB	34KB	24KB
sjeng	256KB	336KB	280KB	263KB
mcf	48KB	144KB	78KB	54KB
lbm	48KB	144KB	77KB	55KB
bzip2	176KB	272KB	202KB	184KB
h264ref	1.51MB	1.53MB	1.52MB	1.52MB

6 讨论与局限

本文继续探索了机器学习算法在代码混淆中的应用,选取了随机森林算法来混淆程序的路径分支,有如下两个优点.

(1)本文扩展了路径分支条件的适用范围,解决了诸如 XOR、AND 等非线性条件的混淆. 对于这类条件,已有的两种方法都需要训练与维数相同个数的分类器,从而会造成较大的额外开销. 相比之下,本文方法对高维数据具有较强的处理能力,在时间开销上优于这两种方法.

(2)本文在对符号执行和约束求解保持了良好的抵抗力的基础上,同时显著地增加了二分查找攻击和暴力破解攻击的难度,具有更好的安全性.

局限:在执行效率方面,本文提出的随机森林路径混淆方法仍可进行优化. 由于随机森林具有并行化执行的特性,可以考虑对预测过程进行并行化处理,进一步减少时间开销.

7 结论

程序在运行时会动态地泄露路径信息,逆向工程可以收集这些信息来理解程序内部逻辑. 为了缓解路径信息泄露,提出了一种基于随机森林的路径混淆方法. 随机森林可以被视为一种黑盒,分类过程与路径分

支的行为相似. 本文利用特殊训练的随机森林替代路径分支,将逆向分析的难度等价于抽取随机森林内部规则的难度. 通过实验评估,证明了本文提出的路径混淆方法具有足够的安全性和实用性.

参考文献

[1] COLLBERG C,THOMBORSON C,LOW D. A Taxonomy of Obfuscating Transformations[R]. Technical report 148, Department of Computer Sciences,The University of Auckland,1997.

[2] BARAK B,GOLDREICH O,IMPAGLIAZZO R et al. On the (Im) possibility of obfuscating programs[A]. Lecture Notes in Computer Science (LNCS)[C]. Berlin:Springer,2001. 1 – 18.

[3] BEAUCAMPS P,FILIOL E. On the possibility of practically obfuscating programs towards a unified perspective of code protection[J]. Journal in Computer Virology,2007,3(1):3 – 21.

[4] FALCARIN P,COLLBERG C,ATALLAH M,et al. Guest editors' introduction: Software protection [J]. Software, 2011,28(2):24 – 27.

[5] COLLBERG C, CLARK T, DOUGLAS L. Obfuscation Techniques for Enhancing Software Security[P]. US:Patent No. 6,668,325,2003-12-23.

[6] KRUEGEL C,ROBERTSON W,VALEUR F. Static disassembly of obfuscated binaries [A]. Proceedings of the USENIX-Security[C]. US:USENIX,2004. 18 – 31.

[7] WANG C,DAVIDSON J,HILL J,et al. Protection of software-based survivability mechanisms[A]. Proceedings of Dependable Systems and Networks[C]. Piscataway, NJ: IEEE,2001. 193 – 202.

[8] CHIPOUNOV V,KUZNETSOV V,CANDEA G. The S2E platform: Design, implementation, and applications [J]. ACM Transactions on Computer Systems (TOCS),2012, 30(1):1 – 49.

[9] KING,JAMES C. Symbolic execution and program testing [J]. Communications of the ACM, 1976, 19 (7): 385 – 394.

[10] CADAR C, KOUSHIK S. Symbolic execution for software testing: three decades later[J]. Communications of the ACM,2013,56(2):82 – 90.

[11] 王颖,谷利泽,杨义先,等. EWFT:基于程序执行过程的白盒测试工具[J]. 电子学报,2014,42(10):2016 – 2023.
WANG Ying,GU Li-ze,YANG Yi-xian,et al. EWFT:execution-based whitebox fuzzing for executables[J]. Acta Electronica Sinica,2014,42(10):2016 – 2023. (in Chinese)

[12] GODEFROID P, LEVIN Y, MOLNAR A. Automated whitebox fuzz testing[A]. Proceedings of Network and Distributed System Security Symposium (Vol. 8)[C]. US:ISOC,2008,151 – 166.

[13] BUGRARA S,ENGLER R. Redundant state detection for dynamic symbolic execution [A]. Proceedings of the USENIX Annual Technical Conference[C]. US: USENIX,2013. 199 – 211.

[14] BRUMLEY D,HARTWIG C,LIANG Z,et al. Automatically Identifying Trigger-Based Behavior in Malware [M]. Botnet Detection,2008. 65 – 88.

[15] SHARIF M,LANZI A,GIFFIN J,et al. Automatic reverse engineering of malware emulators[A]. Proceedings of the 30th IEEE Symposium on Security and Privacy[C]. Piscataway,NJ:IEEE,2009. 94 – 109.

[16] 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.
WANG Z,JIA C-F,LU K. Malicious hidden-code extracting based on environment-sensitive analysis[J]. Chinese Journal of Computers,2012,35(4):693 – 702. (in Chinese)

[17] ZENG J,FU Y,MILLER K,et al. Obfuscation resilient binary code reuse through trace-oriented programming [A]. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security[C]. New York: ACM,2013. 487 – 498.

[18] QU X,ROBINSON B. A case study of concolic testing tools and their limitations[A]. Proceedings of the International Symposium on Empirical Software Engineering and Measurement [C]. Piscataway, NJ: IEEE, 2011. 117 – 126.

[19] SHARIF M,LANZI A,GIFFIN T,et al. Impeding malware analysis using conditional code obfuscation [A]. Proceedings of Network and Distributed System Security Symposium[C]. US:ISOC,2008. 321 – 333.

[20] 王志,贾春福,刘伟杰,等. 一种抵抗符号执行的路径分支混淆技术[J]. 电子学报,2015,43(5):870 – 878.
WANG Zhi,JIA Chun-fu,LIU Wei-jie,et al. Branch obfuscation to combat symbolic execution [J]. Acta Electronica Sinica,2015,43(5):870 – 878. (in Chinese)

[21] WANG Z,MING J,JIA C,et al. Linear obfuscation to combat symbolic execution[A]. Proceedings of the European Symposium Research in Computer Security [C]. Berlin Heidelberg:Springer,2011. 210 – 226.

[22] ZONG N,JIA C. Branch obfuscation using "black boxes" [A]. Proceedings of the Theoretical Aspects of Software Engineering Conference[C]. Piscataway,NJ:IEEE,2014. 114 – 121.

[23] MA H,MA X,LIU W,et al. Control flow obfuscation using neural network to fight concolic testing[A]. Proceed-

ings of the 10th International Conference on Security and Privacy in Communication Networks[C]. Berlin Heidelberg:Springer,2014. 287 – 304.

[24] BREIMAN L. Random forests[J]. Machine learning, 2011,45(1):5 – 32.

[25] 魏静明,李应. 利用抗噪纹理特征的快速鸟鸣声识别[J]. 电子学报,2015,43(1):185 – 190.
WEI Jing-ming,LI Ying. Rapid bird sound recognition using anti-noise texture features[J]. Acta Electronica Sinica,2015,43(1):185 – 190. (in Chinese)

[26] LIU S,PATEL Y,DAGA R et al. Combined rule extraction and feature elimination in supervised classification [J]. IEEE Transactions on NanoBioscience,2012,11(3): 228 – 236.

[27] CRAVEN M, SHAVLIK J. Rule Extraction: Where Do We Go From Here[M]. [S. 1.]:University of Wisconsin Machine Learning Research Group Working Paper, 1999. 1 – 4.

[28] BREIMAN L. Looking Inside the Black Box (Wald Lecture II,Department of Statistics)[R]. US:California University,2002.

[29] SVETNIK V,LIAW A,TONG C,et al. Random forest:A classification and regression tool for compound classification and QSAR modeling[J]. Journal of Chemical Information and Computer Sciences, 2003, 43 (6): 1947 – 1958.

[30] EVANS J S,CUSHMAN S A. Gradient modeling of conifer species using random forests[J]. Landscape Ecology, 2009,24(5):673 – 683.

[31] PAWLAK Z. Rough sets[J]. International Journal of Computer and Information Sciences, 1982, 11 (5): 341 – 356.

[32] JELONEK J,KRAWIEC K,SLOWI R. Rough set reduction of attributes and their domains for neural networks [J]. Computational Intelligence,1995,11(2):339 – 347.

[33] LEE,B. LibRF:A Library for Random Forests[OL]. http://mtv. ece. ucsb. edu/benlee/librf. html,2007.

[34] 王健康,王念平. 一类广义 Feistel 密码安全性能的进一步评估[J]. 电子学报,2013,41(10):1944 – 1947.
WANG Jian-kang, WANG Nian-ping. Further security evaluation for a class of generalized feistel ciphers[J]. Acta Electronica Sinica, 2013, 41 (10): 1944 – 1947. (in Chinese)

[35] OSHIRO T M,PEREZ P S,BARANAUSKAS J A. How many trees in a random forest? [A]. Machine Learning and Data Mining in Pattern Recognition [M]. Berlin: Springer,2012. 154 – 168.

[36] LATINNE P,DEBEIR O,DECAESTECKER C. Limiting the number of trees in random forests[A]. Proceedings of the International Workshop on Multiple Classifier Systems [C]. London UK:Springer-Verlag,2001. 178 – 187.

作者简介

陈 喆 男,1989 年 8 月出生,天津人. 博士生,主要研究方向为软件保护技术.

贾春福(通信作者) 男,1967 年 5 月出生,河北文安人. 教授,博士生导师,主要研究方向为计算机网络与信息安全、可信计算、恶意代码分析.
E-mail:cfjia@ nankai. edu. cn