第 36 卷 第 9 期 计 算 机 工 程 2010 年 5 月
Vol.36 No.9 Computer Engineering May 2010
·安全技术· 文章编号：1000—3428(2010)09—0139—03 文献标识码：A 中图分类号：TP309.2

基于关键应用编程接口图的恶意代码检测

白莉莉，庞建民，张一弛，岳 峰
(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：针对基于特征码的恶意代码检测方法无法应对混淆变形技术的问题，提出基于关键应用编程接口(API)图的检测方法。通过提取恶意代码控制流图中含关键 API 调用的节点，将恶意行为抽象成关键 API 图，采用子图匹配的方法判定可疑程序的恶意度。实验结果证明，该方法能有效检测恶意代码变体，漏报率较低。

关键词：控制流图；关键应用编程接口图；恶意代码检测

Malware Detection Based on Critical Application Programming Interface Graph

BAI Li-li, PANG Jian-min, ZHANG Yi-chi, YUE Feng
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】Aiming at the problem that malware detection method based on signature can be easily subverted by obfuscation techniques, this paper proposes a detection method based on Critical Application Programming Interface Graph(CAG). By statically extracting nodes with critical API calling from Control Flow Graph(CFG) for each malware, each malicious behavior can be presented by one CAG. A matching algorithm based on CAG is used to determine whether a suspicious executable programming has the same malicious behavior as a malware does. Experimental results show that the method can detect malware variants efficiently with low false negative rate.

【Key words】Control Flow Graph(CFG); Critical Application Programming Interface Graph(CAG); malware detection

1 概述

传统的特征码检测以其快速准确识别已知恶意代码的优点被广泛地应用在各种恶意代码检测工具中。为了逃避此类检测，恶意代码编写者们采用了各种技术对原有恶意代码进行变形、混淆[1]，产生大量变种。如何应对混淆变形技术、有效检测恶意代码变种是目前亟待解决的问题。为此，本文提出一种基于关键应用编程接口图(Critical Application Programming Interface Graph, CAG)的恶意程序检测方法。

控制流图(Control Flow Graph, CFG)是描述程序控制流的一种图示方法，它反映了程序中语句的执行顺序和函数之间的相互调用关系。本文方法从控制流图中抽取恶意行为的特征，通过对可执行文件进行静态分析，构建程序的控制流图，在此基础上提取程序的 CAG。对 CAG 进行分析，检查其是否为恶意行为库中定义的行为，最后将程序具有的恶意行为汇总分析，形成可疑程序分析报告。实验结果证明本方法能够有效检测经过变形、混淆之后的病毒程序。

2 恶意代码检测系统

不同版本的恶意代码在控制流图层次上很难改变其特征[2]，因此，本文讨论从恶意代码的行为和控制流上抽取恶意代码的特征。为了证明这种方法的有效性，介绍了一种基于控制流图的恶意代码检测系统框架。其针对的是 Windows 平台 PE(Portable Executable)格式文件，主要分为 CFG 生成、CAG 提取、恶意行为检测 3 个部分，如图 1 所示。为了防止静态分析，恶意程序编写者往往使用加壳、压缩工具，对恶意程序进行修改。如果可疑程序经过第三方压缩软件(如 USP Shell)的处理，则先使用相应的脱壳软件进行脱壳。

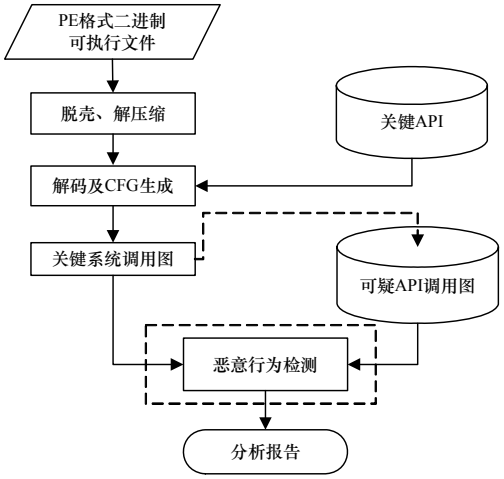


图 1 恶意代码检测系统框架

2.1 恶意行为库的构建

在 Windows 平台上，使用 API 作为系统调用的接口，几乎所有的程序都会调用 API 中的函数来实现相应的功能[3]。无论是无用指令插入还是跳转混淆，都不能改变关键库函数的调用关系[4]。通过分析程序调用的库函数来了解程序的行

基金项目：国家“863”计划基金资助项目(2006AA01Z408, 2009AA01Z434)

作者简介：白莉莉(1985－)，女，硕士研究生，主研方向：信息安全；庞建民，教授；张一弛、岳 峰，硕士研究生

收稿日期：2009-03-24 **E-mail：**lili_b85@sina.com

为，将程序中含有的恶意行为抽象到库函数调用层次上，就能够对抗多种混淆手段。

本文将过程内的 CFG 抽取为一个 API 图。在该图中，CFG 中所有的计算节点都被忽略，只有调用 API 函数的节点被保留。然后从 API 图中抽取 CAG，这里被调用的 API 函数都是对安全具有影响的函数，该 CAG 是在检测阶段使用的。

恶意行为库(Malicious Behavior Database, MBDB)中存放恶意行为的 CAGm。通过对知名的病毒进行分析，抽取恶意程序的 CAG。对得到的 CAG 按照其完成的功能进行划分，形成 CAGm，每一个 CAGm 表示了一个恶意行为的 API 调用关系，并满足以下假设。

假设 1 在一个 CAGm 中，相同的 API 函数只能出现一次，即定义的每个行为中每个 API 函数只能调用一次。

恶意程序非常注重资源的消耗情况，基本不会为了完成某种行为调用相同的函数多次，因此，本文在构建 MBDB 时要求 CAGm 中相同的 API 函数只出现一次，方便存储，且检测效率更高。

假设 2 在一个 CAGm 中，不存在孤立节点，即所有的节点的度数要大于等于 1。

CAGm 是从 CAG 中抽取并划分而得的，CAG 不存在孤立节点，则 CAGm 不会存在孤立节点。

2.2 CAG检测算法

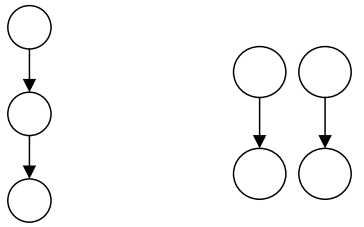
在生成可执行程序 CAG 之后，以 CAG 为基础进行分析，检测该可执行程序中是否含有恶意行为。对生成的 CFG 按照过程(procedure)进行划分，CAG 检测算法也是针对过程进行检测的。

一个过程往往完成某个相对独立、完整的功能。要实现恶意行为，就必须进行一系列操作，这些操作一般是不能分割的，具有原子性和局部性的特点。所以，检测过程内是否含有恶意行为是 CAG 检测的关键。

CAG 检测算法根据恶意行为库中定义的恶意行为，在可疑程序生成的 CAG 中寻找是否有匹配的节点和连通关系。对程序的行为检测可以简化为图中的子图匹配问题。由于子图匹配是 NP 完全问题，本文要处理的只是一类比较特殊的问题，因此可以将问题简化为在 CAG 中寻找对应节点是否存在连通关系。

MBDB 中将一个恶意行为抽象成 CAGm，CAGm=<N, E>，其中，N 是节点，用函数名称进行标示。因为定义的 CAGm 不存在孤立节点，所以对图中的边 Eij: <i,j>进行存储即可完成 CAGm 的存储，一个 CAGm 在库中表示为边集 Se。

检测 CAG 中的每个过程是否存在 Se，每个 Se 包含若干条边 Eij: <i,j>。首先检查一个过程内的 CAG(Gp)是否存在 Eij 这样的边，即判断包含函数 i 的基本块(Bi)和包含函数 j 的基本块(Bj)是否存在调用关系。这种调用可能不是直接的，在它们的连通路径上可能存在其他函数，这样，问题就转化为判断 Bi 和 Bj 是否可达，同时生成 2 个集合 I_List 和 J_List，其中，I_List 是 Gp 中所有能到达 Bj 的 Bi 集合；J_List 是 I_List 能够到达的所有 Bj 的集合。对 Se 集合中的所有元素进行检测，判断 CAGm 的所有边在 Gp 中是否有对应的连通关系，当满足这些连通关系之后，需要进一步验证边与边之间的关系，以确认是否检测出恶意行为 CAGm。但是在每条边都存在的情况下，仍然有不连通的情况，如图 2 所示。



(a)定义的 CAGm　　(b)检测出的 CAG

图 2　CAG 所有边存在但不连通的情况

不难看出，定义的 CAGm 和检测出的 CAG，在 B 点存在 2 种情况：CAGm 的 B 点可以到达 C 点，A 点能够到达 B 点；而检测出的 CAG 中，A 点能够到达 B2 点，B1 点能够到达 C 点，而 B1 和 B2 不重合，使得 A 点无法到达 C 点。虽然在 CAG 中检测出 CAGm 中的 2 条边<A,B>, <B,C>，但是 B 节点不是同一个，不满足 CAGm 的定义。因此，需要对 B 点进行确认。在 CAG 中对所有边进行确认之后，每条边生成 2 个集合 I_list 和 J_list，对这些产生的集合按照函数名称进行分类，CAGm 中有多少函数，就分成多少类。对每一类中的所有集合取交集，若不为空，表明存在这样的 B 点。若所有类中元素的交集都不为空，表明检测出的 CAG 符合 CAGm 的定义，即检测出该恶意行为。这种方法同样适用于图 3 所示的 2 种情况。

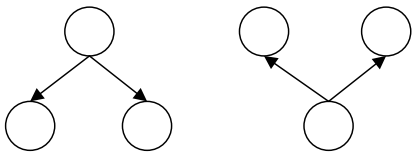


图 3　CAG 所有边存在但不连通的其他 2 种情况

算法 1 判断过程 P 中是否含有边 Eij，生成 I_List 和对应的 J_List

输入 过程 P 的关键系统调用图 Gp，恶意行为 CAG 的边 Eij: <i,j>

输出 Te(表示是否存在边 Eij)，i 节点的集合 I_List，i 节点能够到达的所有 j 节点的集合 J_List

```
I_List:= ∅ ;
J_List:= ∅ ;
Te:=false
for each B ∈ {B|B is one of node of Gp}
do
    if B=i
    then
    I_List=I_List∪B;
    endif
endfor   //得到过程 P 中所有的 i 节点
for each i ∈ I_List do
     O_List:= ∅ ;
     O_List := O_List∪i.outedge;
     for each o ∈ {o|o is one of the O_List}
     do
         if o=j
         then
             J_List:= J_List∪o;
             //得到 i 节点能够到达的所有 j 节点
         Endif
         O_List := O_List∪i.outedge;
```

```
        Endfor
        if J_List= ∅
        then
            I_List= I_List-i;//去掉没有连通 j 节点的 i 节点
        endif
    endfor
    if J_List≠ ∅
    then    Te :=True;
    endif
    Output Te, I_List, J_List
```

算法 2 判断 CAGm 中的所有边是否全部找到

输入 CAG 中每条边的 Eij 的 Te

输出 M(表示 CAGm 中所有边是否全部找到)

```
M :=True //初始值为真
for each Te
do
    if Te =False
        M:=False;
    endif
endfor
Output M
```

利用算法 1 能够得到一个恶意行为的每条边、在 CAG 中 2 个节点集合的 I_List 和 J_List。算法 3 对恶意行为的完整性做出确认，避免出现因为一个过程中出现同名函数而导致的图 2(b)的情况出现。

算法 3 判断一个过程内是否含有一个恶意行为 CAGm

输入 恶意行为 CAGm 中的边集 E 和节点集 N

输出 T(当前过程内是否具有该恶意行为)

```
T:= True; A:= ∅
for each a ∈ N and b ∈ N and c ∈ N and Eab ∈ E and Ebc ∈ E
do
//对 CAGm 中所有相邻的边进行验证
     A= Eab .J_list∩Ebc .I_list;
     //判断具有相同函数名的节点是否是同一节点
     if A:= ∅  then
        T:= False
        Output T
    endif
endfor
Output T
```

连通关系在过程内都可以找到就认为本过程含有该恶意行为。在检测完所有过程中是否含有 MBDB 中定义的恶意行为之后，对检测出的行为进行汇总，去掉重复的行为，得到程序的行为报告。根据每种行为的恶意度计算被检测程序的可疑程度，综合判断被检测程序是否为恶意程序。

3 实验结果

目前一些 Win32 PE 格式的病毒经常用来做分析研究，例如：Doser, Beagle, Fosforo, Hortiga。对于每一个病毒，本文通过文献[1]中的混淆技术进行混淆，产生多个病毒变种。例如，DoserV1 和 Beaglev1 改变数据段；DoserV2 和 Beaglev2 改变控制流实现；FosforoV1 插入死亡代码。使用 7 种不同的病毒扫描软件和本文的检测器对这些病毒的变体进行实验，结果如表 1 所示，其中，K 代表 Kaspersky；N 代表 Norton；M 代表 McAfee；C 代表 ClamAV；R 代表瑞星；J 代表 KV3000；K 代表金山毒霸；R 代表本文的检测器。从实验结果可以看出，本文的检测器检测最为准确。另外，对良性的 PE 文件测试的结果显示本文的检测器的误报率也很低。

表 1　对变形病毒的测试结果

病毒	K	N	M	C	R	J	K	R
Doser	√	√	√	√	√	√	√	√
DoserV1	×	×	√	×	×	×	×	√
DoserV2	×	×	×	×	×	×	×	√
DoserV3	×	×	×	×	√	×	×	√
DoserV4	√	×	×	×	×	×	×	√
DoserV5	×	×	×	×	×	×	×	√
DoserV6	×	×	×	×	×	×	×	√
Beagle	√	√	√	√	√	√	√	√
BeagleV1	√	×	√	×	√	×	×	√
BeagleV2	√	×	×	×	√	×	×	√
BeagleV3	×	×	×	×	×	×	×	√
Fosforo	√	√	√	√	√	√	√	√
FosforoV1	√	√	√	×	×	×	×	√
FosforoV2	√	×	×	×	×	×	×	√
Hortiga	√	√	√	√	√	√	√	√
HortigaV1	×	√	×	×	√	×	×	√
HortigaV2	√	√	×	√	×	×	×	√
HortigaV3	×	√	×	×	×	√	×	√
HortigaV4	×	×	×	×	×	×	×	√

4 结束语

本文提出了一种基于控制流图的恶意代码检测方法。该方法通过对可执行程序的反编译，构建程序的控制流图，从中抽取关键 API 调用节点，形成 CAG，通过与恶意行为库中定义的恶意行为进行比较，检测可执行程序中含有的可疑行为，从而判断程序的恶意程度。实验结果证明，由于将恶意行为抽象为较高层次的系统调用，因此本方法具有较高的检测率、较低的误报率，能够对抗垃圾数据插入、无用代码插入、控制流混淆等多种变形技术。

并非所有恶意代码都是靠 API 调用来实现其恶意行为的，因此，无法保证本文方法对所有恶意代码都适用，但是通过实验可知，本方法在实际应用中是非常有效的。另外，程序的行为不仅取决于调用哪个系统函数，还取决于环境和请求，因此，下一步需要对系统调用的参数进行恢复，进一步提高检测的准确性。

参考文献

[1] Sung A H, Xu Jianyun, Chavez P, et al. Static Analyzer of Vicious Executables(SAVE)[C]//Proc. of the 20th Annual Computer Security Applications Conference. Tucson, USA: [s. n.], 2004.

[2] Kruegel C, Kirda E, Mutz D, et al. Automating Mimicry Attacks Using Static Binary Analysis[C]//Proc. of the 14th USENIX Security Symposium. Baltimore, USA: [s. n.], 2005.

[3] Xu Jianyun, Sung A H, Chavez P, et al. Polymorphic Malicious Executable Scanner by API Sequence Analysis[C]//Proc. of the Conference on Hybrid Intelligent Systems. Kitakyushu, Japan: [s. n.], 2004.

[4] 苏璞睿, 杨　轶. 基于可执行文件静态分析的入侵检测模型[J]. 计算机学报, 2006, 29(9): 1572-1578.

编辑　张　帆