计算机研究与发展
Journal of Computer Research and Development

DOI:10.7544/issn1000-1239.2014.20131897
51(7): 1458-1466, 2014

基于 Dalvik 指令的 Android 恶意代码特征描述及验证

李 挺[1] 董 航[2] 袁春阳[1] 杜跃进[1] 徐国爱[2]

[1](国家计算机网络应急技术处理协调中心 北京 100029)

[2](北京邮电大学信息安全中心 北京 100876)

(liting@cert.org.cn)

Description of Android Malware Feature Based on Dalvik Instructions

Li Ting[1], Dong Hang[2], Yuan Chunyang[1], Du Yuejin[1], and Xu Guo'ai[2]

[1](*National Computer Network Emergency Response Technical Team/Coordination Center of China*, *Beijing* 100029)

[2](*Information Security Center*, *Beijing University of Posts and Telecommunications*, *Beijing* 100876)

Abstract In order to achieve an efficient detection of malicious software on Android, a method to analyze the malware in Android devices using Dalvik instructions has been proposed. The Dalvik executable format (DEX) files are segmented based on its format without decompile. Through the formalize description of Dalvik instructions the features of the program can be simplified and extracted. Using the MOSS algorithm and the Minkowski distance algorithm, it can be determined that whether the current software which will be tested contains malicious code based on the similarity threshold. Finally, a prototype system is built to validate the method with large amounts of random samples. Taking applications which in Android application stores as example, the extraction and description of signatures using this method proves that not only can this static detection method based on Dalvik instructions detect malicious code quickly, but also has a very low rate of false positives and false negatives. Experiments results confirm that the method proposed by this paper is feasible and credible and it is applicable for rapid detection of Android malicious code.

Key words Dalvik instruction; Android; malicious code; formal description; similarity

摘 要 为实现 Android 平台下恶意软件的高效检测,提出了一种基于 Dalvik 指令的 Android 恶意代码特征形式化描述和分析方法,能够在无需反编译应用程序的基础上,快速检测样本的恶意特征.该方法首先依照 DEX 文件格式对 Android 应用程序切分得到以方法为单位的指令块,通过对块中 Dalvik 指令进行形式化描述以实现程序特征的简化和提取,之后综合使用改进的软件相似度度量算法和闵可夫斯基距离算法计算提取特征与已知恶意特征的相似度,并根据相似度比对结果来判定当前待测软件是否含有恶意代码.最后建立原型系统模型来验证上述方法,以大量随机样本进行特征匹配实验.实验结果表明,该方法描述特征准确、检测速度较快,适用于 Android 恶意代码的快速检测.

关键词 Dalvik 指令;安卓;恶意代码;形式化描述;相似度

中图法分类号 TP309

收稿日期:2013-12-16;**修回日期**:2014-03-24

基金项目:国家科技重大专项基金项目(2012ZX03002012);国家信息安全专项基金项目(发改委高技[2012]1424 号)

通信作者:袁春阳(ycy@cert.org.cn)

随着移动互联网的飞速发展,移动智能终端和移动应用的种类、数量呈井喷式增长,移动智能终端已成为我国网民使用数量最多的上网工具.然而,随之而来的网络与信息安全问题日益凸显,移动智能终端安全事件层出不穷,移动恶意应用肆意泛滥,个人隐私窃取、资费恶意消耗等安全问题时有发生,严重影响行业可信度,阻碍行业健康发展.其中最主要的问题是移动恶意代码快速增长、更新频繁带来的安全问题.因此,移动恶意代码特别是 Android 平台的恶意代码检测与分析成为重要的研究问题.

针对恶意代码检测分析问题,桌面平台下的恶意软件分析技术已经相对成熟.然而在移动平台下,由于平台、架构以及指令集迥异,外加大量恶意软件通过代码混淆等手段隐蔽其恶意行为来提升分析难度,因此移动平台下恶意软件分析难度较高.现有的移动平台恶意软件分析方法在行为建模、检测准确性、检测效率、检测能力与系统实用性方面仍需进一步完善.

目前,移动平台的恶意应用特征大部分从应用自身的特征入手,如 Desnos[1] 提出了一种基于相似距离的软件相似度检测以及恶意软件检测系统,从应用程序中的方法中提取特征进行压缩和比较,以此来判定 2 个应用程序相似度.基于语义的检测方法为更好地检测病毒变种提出了可能,目前从语义角度对桌面应用程序特征提取和检测方法都已有部分研究成果[2-3].然而针对 Android 平台 Dalvik 指令集的分析还鲜有人提出.

为了解决 Android 平台恶意代码分析问题,提高分析效率,本文在总结几种现有的智能终端上的应用程序分析方法和工具的基础上,提出了一种基于指令特征的 Android 恶意代码检测方法,用于快速从测试样本中筛选出可疑应用程序.

本文的贡献主要包括如下 2 个部分:

1) 建立了一种基于指令序列的特征描述方法,本方法从指令层次抽象特征,将指令序列之间的关系建立序列特征.该特征可表示一类恶意代码的共同特点,用于快速判断待测样本是否含有恶意代码;

2) 对本文方法进行实验验证,对 ADRD 等实例样本进行特征提取与匹配等实验.实验结果表明,本文所述方法提取的特征具有强代表性,对恶意代码具有较好的识别能力.

1 相关工作

目前恶意软件检测方法主要有基于特征代码(signature-based)的检测方法和基于行为(behavior-based)的检测方法.基于特征代码的检测方法通过检测文件是否拥有已知恶意软件的特征代码(如一段特殊代码或字符串)来判断其是否为恶意软件.基于行为的检测方法则依靠监视程序的行为与已知的恶意行为模式进行匹配,以此判断目标文件是否具备恶意特征.基于行为的分析方法又分为动态分析方法和静态分析方法.动态分析方法是指利用沙盒或虚拟机来模拟运行程序,通过监控或者拦截的方式分析程序运行的行为;静态分析方法则是通过逆向手段抽取程序的特征,分析其中的函数调用序列等.基于特征的检测方法则根据由恶意代码中提取的特征进行检测,与基于行为的检测方法相比,具有效率高、误报率低等优点,因此被广泛应用于恶意代码检测工具之中,是目前恶意代码检测的主流方法.

在恶意代码分析方面,国内外已有大量针对桌面平台或 Web 平台的研究成果.相比之下,智能终端平台上的恶意应用检测还处于初期研究阶段.2004 年和 2005 年,Dagon 等人[4] 以及 Leavitt[5] 揭示手机病毒的出现,才引发了人们的重视,使得移动平台软件安全分析进入研究阶段.2007 年,Cheng 等人[6] 针对智能手机病毒提出了一种病毒检测及预警系统.这些应用和分析主要针对目标是 Nokia 公司的 Symbian 平台.随着近几年手机操作系统的巨大变革,针对智能终端的攻击开始转移阵地,Android 成为黑客攻击的主要目标.

2009 年,Shabtai 等人[7] 从整体上对 Android 系统结构及安全机制进行了分析,并综合对比了几种 Android 安全防范措施的特点.随后许多学者对 Android 系统的安全防护进行了研究:Sanz 等人[8] 试图通过 Android 应用程序权限声明的区别来发现恶意应用程序;Schmidt 等人[9] 将自组织网络(ad hoc)中的分布式计算机制引入 Android 应用分析中.与此同时,Android 平台著名应用分析工具 Androguard[10] 作者将压缩距离(NCD)[11] 引入 Android 的应用程序相似性分析中,即当 2 个应用程序相似,且其中一个程序含有某个已知恶意程序时,表明此应用程序为恶意软件.这种方法在当 2 个应用程序含有相同的恶意代码,却因为实现主体功能的代码不同时,无法检测出恶意软件.同时,目前的静态分析基本都要通过反编译得到源代码或者使用编译后的指令文件再进行后续分析,并没有针对 Dalvik 可执行文件(DEX)进行直接分析.

在针对特征的恶意代码检测中,对特征的准确

描述决定了方法的检测能力和检测效率. 在形式化描述方面,大部分的工作是针对 x86 平台下的汇编指令进行的,鲜有人针对 Dalvik 指令进行形式化描述工作. Androguard 作者使用函数熵等特征对函数进行形式化描述[10],但是此描述针对的是一个函数,没有深入到基本块,也没有对指令等详细内容进行描述,即描述范围太宽泛,仅针对应用的相似性,无法细化到应用程序行为的相似性比较.

在其他 Android 恶意应用分析方面,Felt 等人[12]对 Android 系统的权限进行了分析,即以权限为特征,通过 Android 应用软件的权限来判断是否为恶意软件. 目前基于特征的静态检测很多都是将应用软件进行反编译得到源代码,然后对源代码进行分析并提取特征,根据提取的特征进行恶意软件的检测.

基于以上分析,本文的主要研究内容是 Android 平台下 Dalvik 指令形式化描述方法和恶意代码特征提取及检测方法. 本文首先给出了一种 Dalvik 指令集的形式化描述方法,之后对恶意样本进行分析、提取并筛选特征,提出了基于形式化描述特征的恶意代码检测方法,最后建立了一个针对 Android 平台下恶意应用程序的分析原型系统,验证了文中所提方法的有效性.

2 Dalvik 指令集形式化描述方法

Dalvik① 是 Google 公司设计的用于 Android 平台的虚拟机. 与 Java 虚拟机不同,Dalvik 虚拟机使用 DEX 格式的文件,并且使用基于寄存器的指令集. 本文基于 Dalvik 指令集的特点,定义了一种基于 Dalvik 指令集的形式化描述语言,用来描述恶意代码的特征.

2.1 形式化特征描述语言

对 Android 应用程序指令特征的形式化描述,首先需要从 Dalvik 指令集中精简出反映程序语义的指令、去除无关指令或干扰项. 去除的无关指令包括寄存器、跳转目的等,这些指令在应用被重新编译时很容易随编译环境而变化. 精简后的指令仅包括函数调用、赋值、跳转等. 由于在实际的分析中,较多恶意应用程序通过 Java 的反射机制调用相关代码,因此在对语义进行提取时暂时忽略调用方法名称.

其次,基于精简后的指令,我们定义了一种形式化特征描述语言,具体定义如图 1 所示:

Procedure := *StatementList*;
StatementList := *StatementList* + *Statement*;
Statement := *M*|*R*|*G*|*I*|*T*|*P*|*V*;
M := MOVE|MOVE_WIDE_FROM16|;
R := RETURN|RETURN_OBJECT;
G := GOTO|GOTO_16|GOTO_32;
I := IF_EQ|IF_NEZ|IF_LEZ;
T := AGET_BOOLEAN|IGET_BOOLEAN|SGET_BOOLEAN;
P := APUT_BOOLEAN|IPUT_BOOLEAN|SPUT_BOOLEAN;
V := INVOKE_INTERFACE_RANGE.

Fig. 1 Definition of formal description.

图 1 形式化特征描述定义

上述形式化描述语言中,*M* 代表 move 一类指令集合,包括 MOVE,MOVE_WIDE_FROM16 等 13 种指令,表示数据移动. 指令后的寄存器相关信息忽略不计.

R 代表 return 一类指令集合,该指令集合包括 RETURN,RETURN_OBJECT 等 4 种返回指令,表示函数返回类型.

G 代表 goto 一类指令集合,该指令集合包括 GOTO,GOTO_16 和 GOTO_32 这 3 类跳转指令,表示从当前地址强行跳转到另一个地址,并执行另一个地址的指令.

I 代表 if 一类指令集合,该指令集合包括 IF_EQ,IF_NEZ,IF_LEZ 等 12 种条件判断指令,表示条件判断语句,并根据判断的条件决定是否跳转. 为方便后续计算和分析,在 *I* 之前将函数模块进行切分,得到更加微小的模块指令序列,即以"]@[*I*"替换"*I*",其中@作为分块间的分隔符.

T 代表 aget,iget 和 sget 这 3 类指令合成的指令集合,该指令集合包括 AGET_BOOLEAN,IGET_OBJECT,SGET_SHORT 等 30 种赋值指令,表示赋值语句,将一个已知或者计算后的结果赋值给当前地址.

P 代表 aput,iput 和 sput 这 3 类指令合成的指令集合,该指令集合包括 APUT_BOOLEAN,IPUT_OBJECT,SPUT_SHORT 等 30 种赋值指令,表示将一个已知数值(整数型或者 char 型等)赋值到一个数组中,并加入索引值.

V 代表 invoke 一类指令集合,该指令集合包括 INVOKE_DIRECT,INVOKE_VIRTUAL 等 15 种调用指令,表示调用接口方法等一些函数调用命令.

将原始 Dalvik 指令精简并形式化描述后,可以

① http://source.android.com/devices/tech/dalvik

将复杂指令特征以简单序列表示.

2.2　特征描述举例

图 2 给出一段具有恶意行为的代码片段. 经分析可知,此代码中存在后台下载恶意应用软件等恶意行为. 使用 IDA 对包含此段恶意代码的 DEX 文件进行分析,可以得到其 Dalvik 指令序列如图 3 所示.

```
HttpResponse localHttpResponse = localDefaultHttpClient.execute(localHttpP
if (localHttpResponse.getStatusLine().getStatusCode() != 200)
  return false;
InputStream localInputStream = localHttpResponse.getEntity().getContent();
File localFile = new File(savefilepath + "myupdate.apk");
FileOutputStream localFileOutputStream = new FileOutputStream(localFile);
byte[] arrayOfByte = new byte[1024];
```

Fig. 2　Malicious code sample.

图 2　恶意行为实现代码

```
Method 1686 (0x696):
private boolean
com.xxx.yyy.UpdateHelper.GetO(
  java.lang.String p0)
                        # CODE XREF:
new-instance                    v9,
invoke-direct                   {v9}
move-object/from16              v0,
iget                            v0,
move/from16                     v16,
if-nez                          v16,
new-instance                    v13,
const-string                    v16,
const/16                        v17,
const-string                    v18,
move-object                     v0,
move-object/from16              v1,
move/from16                     v2,
move-object/from16              v3,
invoke-direct                   {v0,
invoke-virtual                  {v9}
move-result-object              v16
const-string                    v17,
move-object/from16              v0,
move-object/from16              v1,
move-object                     v2,
invoke-interface                {v0,

                        # CODE XREF:
new-instance                    v11,
move-object                     v0,
move-object/from16              v1,
invoke-direct                   {v0,
const-string                    v16,
const-string                    v17,
move-object                     v0, v
move-object/from16              v1, v
move-object/from16              v2, v
invoke-virtual                  {v0,
const-string                    v16,
const-string                    v17,
move-object                     v0, v
move-object/from16              v1, v
move-object/from16              v2, v
invoke-virtual                  {v0,
x2A07A:
invoke-virtual                  {v9,
move-result-object              v14
invoke-interface                {v14}
move-result-object              v16
invoke-interface/range          {v16.
move-result                     v15
const/16                        v16,
move                            v0, v
move/from16                     v1, v
if-eq                           v0, v
const/16                        v16,

                                # CODE XREF:
                                # UpdateHelpe
return                          v16
```

Fig. 3　Malicious code analysis by IDA.

图 3　IDA 恶意代码分析图

根据形式化描述语言定义,可得到图 3 所示代码片段的形式化描述语句为[*VMTM*]@[*IMMMMVVMMMMVMMVMMMVMMMVMMMVVMVMVMMM*]@[*IRVMVMTVMVVMVMMMVVMMVMMM*]@[*IVVVMVGMMMMVGMMVGMMVG*].

由此可知,经过形式化描述后的指令序列简化了大量无关指令,仅保留了关键的指令序列. 这种简化和存储方式有利于进一步对指令序列进行相似度比对与分析.

3　特征提取与分析

基于本文对 Dalvik 指令的形式化描述方法,可以提取恶意应用程序关键方法的形式化描述特征,并以此特征为依据进行分析,主要包括以下 3 个步骤:

1) 特征提取. 对给定恶意软件进行反编译,得到源代码. 对源代码进行分析,得到源代码中实现恶意行为的代码片段.

2) 特征描述. 对具有恶意特征的代码片段进行形式化描述,得到对应的指令序列并记为备选特征.

3) 特征筛选. 通过实验,从备选特征剔除掉正常软件具有的特征,得到对检测这一族恶意软件的有效特征,记为最终特征.

3.1　特征提取与描述

基于指令的恶意代码特征提取过程是指在获得恶意软件样本之后,通过对其进行反编译得到的源代码进行人工深度分析,得到恶意代码对应的初步指令序列,再进行筛选剔除非代表性特征.

以某含有 ADRD 的恶意应用程序为例,在对其进行反编译后获取的源代码分析,发现其实现恶意行为的主要代码来源于 UpdateHelper.java 文件,图 4 展示了此文件中的部分恶意代码:

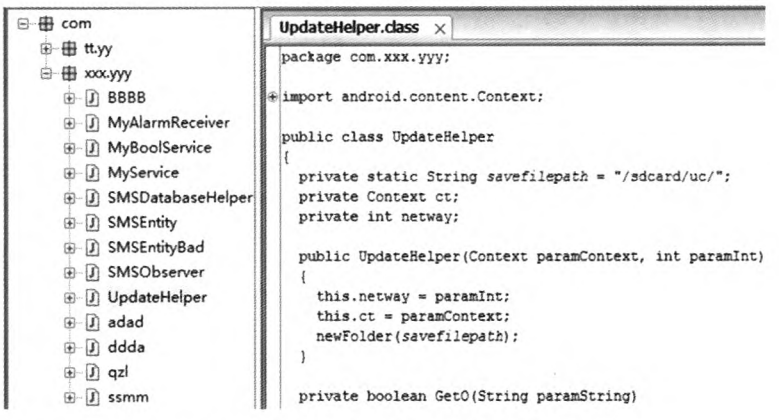

Fig. 4 Malicious code in ADRD sample.

图 4 ADRD 部分恶意代码

通过对包含恶意代码的 DEX 文件进行指令解析并进行形式化描述,可以得到初步的恶意特征. 例如,可得到 ADRD 类恶意应用程序的部分特征为 $[VMVMPVMPT]@[IPT]@[IPVMPVVMPTTVT]@[IVVPVMVMTVR]@[VMVMVMP]@[IVM]@[EVMVM]@[IPTTTTMVVVMTTVR]@[VPVMPVPTVMPTVMPTVMPPR]$.

3.2 特征分析算法

为了判断测试代码是否与某段恶意代码相似,需要分析对应的指令序列和特征的相似度,其本质是判断 2 段经形式化描述后的代码是否相似,最终可以归结为判断 2 个字符串向量的相似度. 本文采用 MOSS(measure of software similarity)算法、闵可夫斯基距离以及编辑距离算法进行相似度度量.

3.2.1 MOSS 算法

本文将 MOSS 算法改进以更符合我们实验的要求. 本文的改进 MOSS 算法如下.

算法 1. 用于恶意代码检测的改进 MOSS 算法.

输入:待测样本 SW. 特征集合 P,滑动窗口大小 k;

输出:特征命中数量.

其中,方法 $DA()$ 表示将样本进行切分并进行形式化描述.

1) 将 P 以][为分隔符划分为 N 块,则有 $P=(p_1,p_2,\cdots,p_N)$. 在 P 上移动一个大小为 k 滑动窗口,构造出一个由 $v=N-k+1$ 个子串组成的列表 $shingles[v]$,令这个操作为 $Kgram()$.

2) 将 SW 中的每个方法 i 分别通过步骤 2)~4)进行处理,得到经过形式化描述后的指令特征,即

$$method[i]\leftarrow SW.DA().$$

3) 令 $k=2$,将 $method[i]$ 中的每个元素按照步骤 1)进行分块,得到方法 i 的特征序列 $C_i[u]$,即

$$C_i[u]\leftarrow Kgram(method[i]).$$

4) 计算从每个方法中提取的特征与恶意特征的相似度,其中使用的闵可夫斯基距离和编辑距离算法 $Levenshtein$ 将在后文具体描述,即

for each u, v do

$Length[u\times v]=Levenshtein(shingles[v], C_i[u])$;

end for

$$\rho(P,method[i])=\left[\sum_{x=1}^{u\times v}|Length[x]|^K\right]^{\frac{1}{K}}.$$

5) 对于每个被检查的样本 SW,构造 $pairwise$,记录 SW 中所有方法 i 命中的特征数量和,即

$pairwise=0$;

for each i do

if ($\rho(P,method[i])>threshold$)

$pairwise+1$.

end if

end for

6) 当特征的命中率大于阈值 $percent$ 时,表明此软件中含有恶意代码,即

if ($pairwise/P.length)>percent$

successfully detect malicious code.

end if

3.2.2 闵可夫斯基距离及编辑距离算法

在 MOSS 算法中,需要计算特征之间的相似度,本文使用字符串之间的距离来进行评估. 本文使用编辑距离算法计算 2 个不等长字符串之间的距离. 编辑距离是指 2 个字符串之间,由一个转成另一个所需的最少编辑操作次数,算法如下.

算法 2. 编辑距离算法 *Levenshtein*.

输入:字符串 str_1 和 str_2;

输出:2 个字符串之间的距离 D.

1) 若 str_1 或 str_2 的长度为 0,则返回另一个字符串的长度,即

$m=str_1.length()$;

$n=str_2.length()$;

if (min(m,n)==0)

  return D=max(m,n);

end if.

2) 初始化(n+1)×(m+1)的矩阵 $\boldsymbol{d}$,并使第 1 行和第 1 列的值从 0 开始增长.

3) 扫描 2 字符串(n×m 级),即

for i=1 to m do

  for j=1 to n do

    if ($str_1[i]$==$str_2[i]$)

      $temp$=0;

    else $temp$=1;

    $\boldsymbol{d}[i,j]$=min($\boldsymbol{d}[i-1,j]$+1,$\boldsymbol{d}[i,j-1]$+1,$\boldsymbol{d}[i-1,j-1]$+$temp$)

    end if;

  end for

end for.

4) 返回 str_1 和 str_2 的编辑距离 D,即

$$D=\boldsymbol{d}[n][m].$$

算法 1 中需要使用闵可夫斯基距离算法计算 2 个滑动窗口 M 和 N 之间的相似度:

$$\rho(M,N)=\left[\sum_{i=1}^{k}|Levenshtein(M,N)|^{K}\right]^{\frac{1}{K}},$$

其中,字符串 M 和 N 之间的距离由编辑距离计算,k 为 MOSS 算法中滑动窗口的长度,K 为常值参数,取值由具体场景而定.依据经验,本文中 K 取值为 2.

3.3　特征筛选

通过特征提取所选取的特征不一定能很好地满足检测需求,在特征提取的基础上需要进一步明确这些特征的有效性和试用范围,最终从这些特征中提取出有效的恶意代码特征.

以上文 ADRD 类特征为例,根据 MOSS 算法,使用由人工分析提出的 16 个备选特征对已知 50 个 ADRD 类恶意样本及 50 个正常样本进行特征筛选.

图 5 展示了 ADRD 类恶意样本的特征比对结果.可见,除了编号为 4,5,10 之外的 13 个特征同时对正常样本和恶意样本具有较高命中率,说明这些特征无法有效检测 ADRD 类恶意代码,将被剔除.相反,编号为 4,5,10 的特征几乎没有命中正常样本,同时可对恶意样本具有高命中率,因此最终选取这 3 个特征为 ADRD 类恶意样本的最终特征.

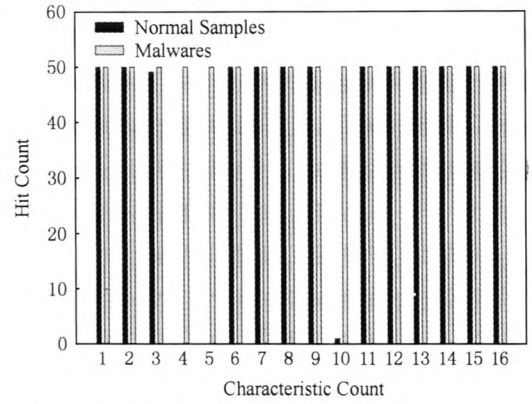

Fig. 5　Distribution of feature-hitted-samples.

图 5　特征命中样本分布图

4　系统设计及实验结果分析

4.1　系统设计

基于本文方法我们研发了一套原型系统,主要包括特征提取和检测 2 部分,如图 6 所示:

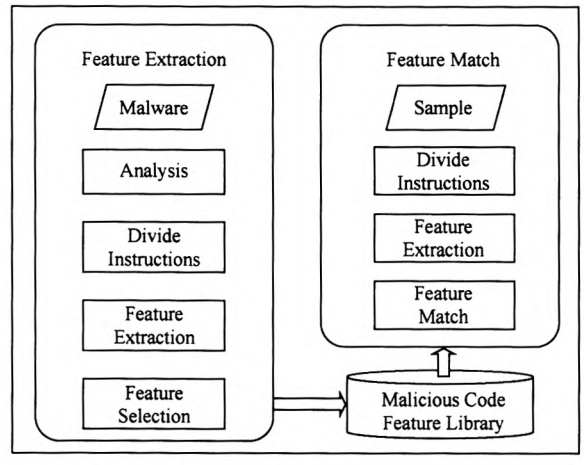

Fig. 6　Framework of prototype system.

图 6　原型系统框架图

在特征提取阶段,首先将恶意样本进行反编译,查看到恶意代码部分,从中提取特征经形式化描述得到初步的恶意特征序列;其次使用大量同类样本对备选特征进行匹配和优化,将其中代表性不强的特征删除,得到最终的恶意特征库.在特征匹配阶段,将输入的样本根据 Dalvik 格式直接将 DEX 文

件按照方法进行切分,得到方法的形式化描述特征序列,再与特征库进行比对,得到匹配结果.

4.2 实验结果及分析

为了测试所提取的恶意代码特征对于真实样本的效果,我们在实验中选取一些有代表性的恶意代码及正常样本进行测试,并与第三方杀毒软件结果进行比较.测试的恶意样本集来源于北卡罗来纳州立大学的恶意样本数据库①,正常样本来源于 Google Play 商店随机下载②.在实验中,我们选取单个恶意代码样本提取系列特征,利用此特征序列对小型样本库 A(包括一些已知恶意样本和正常样本)进行测试,并根据此样本对特征库进行优化,然后利用优化后的特征库对大量样本(7 700 个正常样本及 300 个恶意样本)进行测试,并用第三方杀毒软件对样本库 B 进行扫描,将两者结果进行对比.

在本文中,我们以 ADRD 以及 DroidKungFu 为样本的实验结果为例,说明特征提取及检测结果.在实验中,每类恶意代码的有效特征数为 3 个,检测结果中命中 2 个即认为检测样本为恶意软件.

本文定义准确率(true positive rate, TPR)为成功检测出的恶意软件的个数和恶意软件总数之比.本文使用 TP 表示成功被检测出的恶意软件个数,FN 表示被误检测为正常软件的恶意软件个数,则 TPR 计算公式如下:

$$TPR=\frac{TP}{TP+FN}.$$

本文定义误报率(false positive rate, FPR)为被误报为恶意软件的正常软件的个数和正常软件总数之比.本文使用 FP 表示被误报为恶意软件的正常软件个数,TN 表示正确分类的正常软件,则 FPR 计算公式如下:

$$FPR=\frac{FP}{FP+TN}.$$

利用优化后的特征库对大型样本库 B 扫描以及使用杀毒软件(金山毒霸、360 杀毒、KAV 和 Avast)进行查杀进行结果对比.

TPR 和 FPR 对比结果如图 7 和图 8 所示.从图 7 可知,对于样本数为 40 的 ADRD 类,所有的检测工具准确率均为 100%;而对于样本数为 300 的 DroidKungFu 类,本文检测系统与金山、360 杀毒软件、KAV 以及 Avast 结果接近,准确率均高于 98%,而 360 杀毒软件准确率只有 8%,说明 360 杀毒软件依靠的特征库中缺少此类病毒特征,因此准确率较低.这也说明了普通杀毒软件存在的一个问题,即对软件本身的特征库非常依赖,含有恶意代码的软件变化幅度超过一个较小的范围就难以被检测出来.

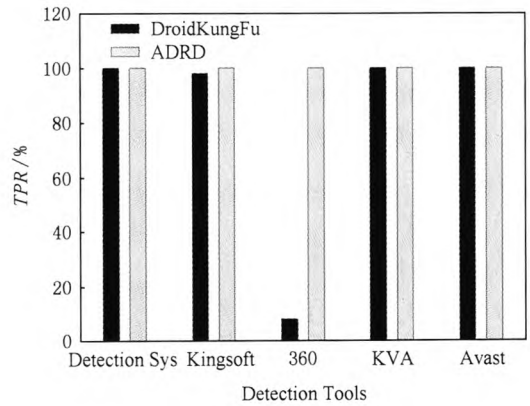

Fig. 7 Result of TPR.

图 7 TPR 对比结果

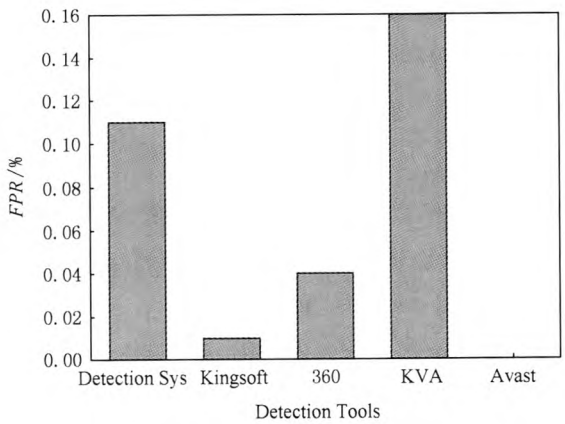

Fig. 8 Result of FPR.

图 8 FPR 对比结果

在使用本文检测系统在对 ADRD 和 DroidKungFu 进行交叉检测时,没有出现相互误报的情况,表明通过特征提取与比对时所提取的特征非常典型,可以顺利地区分出 2 种恶意软件.

从图 8 所示数据可以看出,本文检测系统与其他 4 个杀毒软件结果都比较接近,误报率均低于 1%.

综合图 7 和图 8 的分析结果可以看出,本文的结果是有效的,本文提出的方法是可信且可行的.

本文检测系统、金山毒霸和 360 杀毒软件检测时间如图 9 所示.结果表明,本文系统检测时间效率基本为 24 min 检测 1 000 个样本,平均每个 1.44 s;

① http://www.malgenomeproject.org/policy.html

② https://play.google.com/store

同时金山毒霸检测时间是我们系统的 2 倍;360 杀毒软件时间很短,但是其准确率太低.本文检测系统和 KAV 与 Avast 检测时间和结果比较接近.

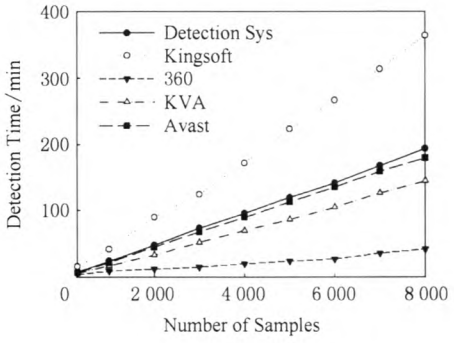

Fig. 9　Comparison of detection time.

图 9　检测时间对比图

使用不同的特征数,对相同的 10 个应用进行检测所用时间如表 1 所示.由表 1 数据进行拟合可以发现,随着特征数的增长,检测时间呈对数增长趋势,大量样本的分析时间趋近于某一稳定值.目前看来这一常值还比较巨大,对于今后大量特征分析匹配,还需要对算法进行优化和改进.

Table 1　Detection Time of Feature Numbers

表 1　特征数目与检测时间的关系

Feature Count	Detection Time/min
5	0.38
10	0.87
20	1.65
40	3.02
80	5.27
160	9.62

根据本节实验结果的分析,在特征数量不大的情况下,本文所提出的方法可以保证较高的命中率、较低的误报率.

4.3　讨　论

通过对多种恶意代码样本进行实验测试,验证了本文提出的基于指令特征的 Android 恶意代码分析方法的有效性.本文提出的特征提取及检测方法对正常程序与恶意代码有较高的辨识度,同时由于检测时并没有对恶意软件进行反编译操作,而是直接对其源文件进行操作,因此检测系统的时间效率也比较理想.但是,目前的代码分析、特征提取以及特征优化中依然包括大量的人工操作,提取的特征数量以及涵盖的恶意代码种类较少,极大地降低了我们检测系统的功能.对于此问题,在接下来的工作中可加入数据挖掘相关内容进行改善.

5　总　　结

本文提出了一种基于 Dalvik 指令特征的 Android 恶意代码形式化描述方法.该方法将 Android 应用程序中的 Dalvik 指令序列进行精简和进行形式化描述,得到对应的初始特征.再通过对比实验和人工分析,对初始特征进行筛选得到程序片段对应的最终特征.该特征可用于 Android 恶意代码的特征描述及恶意代码检测.实验数据表明,该方法提取的特征具有较好的代表性,基于指令特征的检测方法对恶意代码有较高的识别率.

在下一步工作中,需要结合数据挖掘的相关方法,提取出更精准和更多的特征,在保证检测准确率的基础上进一步优化算法,提高检测效率.

参　考　文　献

[1] Desnos A. Android: Static analysis using similarity distance [C] //Proc of the 45th Hawaii Int Conf on System Sciences (HICSS). Los Alamitos, CA: IEEE Computer Society, 2012: 5394-5403

[2] Christodorescu M, Jha S, Seshia S A, et al. Semantics-aware malware detection [C] //Proc of the 2005 IEEE Symp on Security and Privacy (Oakland'05). Los Alamitos, CA: IEEE Computer Society, 2005: 32-46

[3] Wang Rui, Feng Dengguo, Yang Yi, et al. Semantics-based malware behavior signature extraction and detection method [J]. Journal of Software, 2012, 23(2): 378-393 (in Chinese)

(王蕊, 冯登国, 杨轶, 等. 基于语义的恶意代码行为特征提取及检测方法[J]. 软件学报, 2012, 23(2): 378-393)

[4] Dagon D, Martin T, Starner T. Mobile phones as computing devices: The viruses are coming! [J]. IEEE Pervasive Computing, 2004, 3(4): 11-15

[5] Leavitt N. Mobile phones: The next frontier for hackers? [J]. Computer, 2005, 38(4): 20-23

[6] Cheng J, Wong S H, Yang H, et al. Smartsiren: Virus detection and alert for smartphones [C] //Proc of the 5th Int Conf on Mobile Systems, Applications and Services. New York: ACM, 2007: 258-271

[7] Shabtai A, Fledel Y, Kanonov U, et al. Google Android: A state-of-the-art review of security mechanisms [OL]. [2012-12-05]. http://arxiv.org/ftp/arxiv/papers/0912/0912.5101.pdf

[8] Sanz B, Santos I, Laorden C, et al. PUMA: Permission usage to detect malware in android [C] //Proc of the 5th Int Conf on Computational Intelligence in Security for Information Systems (CISIS'02). Berlin: Springer, 2013: 289-298

[9] Schmidt A D, Bye R, Schmidt H G, et al. Static analysis of executables for collaborative malware detection on android [C] //Proc of the 8th IEEE Int Conf on Communications (ICC'09). Piscataway, NJ: IEEE, 2009: 1-5

[10] Desnos A. Androguard: Reverse engineering, malware and goodware analysis of Android applications ... and more (ninja!)[CP/OL]. [2013-03-26]. http://code.google.com/p/androguard/

[11] Cilibrasi R, Vitanyi P M B. Clustering by compression [J]. IEEE Trans on Information Theory, 2005, 51(4): 1523-1545

[12] Felt A P, Chin E, Hanna S, et al. Android permissions demystified [C] //Proc of the 18th ACM Conf on Computer and Communications Security. New York: ACM, 2011: 627-638

Li Ting, born in 1983. PhD, engineer. Member of China Computer Federation. His research interests include mobile Internet security, malware detection, security evaluation, security forensics, etc.

Dong Hang, born in 1986. PhD candidate of Beijing University of Posts and Telecommunications. His research interests include software security of mobile Internet, application vulnerabilities detection.

Yuan Chunyang, born in 1979. PhD, senior engineer. His current research interests include network information security and software system.

Du Yuejin, born in 1972. PhD, professor. His main research interests include cyber security, information security, network simulation, etc.

Xu Guo'ai, born in 1972. PhD, professor of Beijing University of Posts and Telecommunications, China. His current research is software security.

基于Dalvik指令的Android恶意代码特征描述及验证

作者： 李挺， 董航， 袁春阳， 杜跃进， 徐国爱， Li Ting， Dong Hang， Yuan Chunyang， Du Yuejin， Xu Guo'ai

作者单位： 李挺,袁春阳,杜跃进,Li Ting,Yuan Chunyang,Du Yuejin(国家计算机网络应急技术处理协调中心 北京 100029)， 董航,徐国爱,Dong Hang,Xu Guo'ai(北京邮电大学信息安全中心 北京 100876)

刊名： 计算机研究与发展 ISTIC EI PKU

英文刊名： Journal of Computer Research and Development

年，卷(期)： 2014,51(7)

参考文献(15条)

1. 查看详情

2. 查看详情

3. 查看详情

4. Desnos A Android:Static analysis using similarity distance 2012

5. Christodorescu M;Jha S;Seshia S A Semanticsaware malware detection 2005

6. 王蕊;冯登国;杨轶 基于语义的恶意代码行为特征提取及检测方法 201 (02)

7. Dagon D;Martin T;Starner T Mobile phones as computing devices:The viruses are coming! 2004(04)

8. Leavitt N Mobile phones:The next frontier for hackers 2005(04)

9. Cheng J;Wong S H;Yang H Smartsiren:Virus detection and alert for smartphones 2007

10. Shabtai A;Fledel Y;Kanonov U Google Android:A state-of-the-art review of security mechanisms 2012

11. Sanz B;Santos I;Laorden C PUMA:Permission usage to detect malware in android 2013

12. Schmidt A D;Bye R;Schmidt H G Static analysis of executables for collaborative malware detection on android 2009

13. Desnos A Androguard:Reverse engineering,malware and goodware analysis of Android applications...and more (ninja!) 2013

14. Cilibrasi R;Vitanyi P M B Clustering by compression 2005(04)

15. Felt A P;Chin E;Hanna S Android permissions demystified 2011