DOI：10.7652/xjtuxb201510014

采用函数调用关系的注入型 Android 恶意应用检测

王欢[1]，来欢[2]，李国栋[3]，田达[2]，梁博[2]
（1. 西安工程大学服装与设计艺术学院，710048，西安；2. 西安阎良国家航空高技术产业基地管理委员会，710089，西安；3. 西安交通大学网络中心，710049，西安）

摘要：针对注入型 Android 恶意应用日益泛滥、传统检测方法依赖大量已知特征的问题，提出了采用函数调用关系的注入型 Android 恶意应用检测方法。该方法无须依赖大量已知特征，仅通过分析注入型 Android 恶意应用的自身结构特征即可实现对该类恶意应用的有效检测，并能够实现对未知恶意代码家族的识别。所提方法在 smali 代码的基础上构建函数调用关系图，并进一步进行子图划分，通过判定各子图威胁度确定是否存在恶意行为。检测过程无需动态行为分析辅助，因此分析检测时间短、效率高。该方法不仅可以检测出 Android 应用是否存在恶意行为，还可根据子图威胁度确定包含恶意行为的具体代码。经过对 1 260 个 Android 恶意应用和 1 000 个正常应用的实验分析发现：所提方法能够很好地检测注入型 Android 恶意应用，当误报率为 8.90％的时候，检测率达到 95.94％，相对于主流 Android 恶意应用检测系统 Androguard，检测效果有显著提升。

关键词：Android；恶意代码；静态分析；函数调用关系
中图分类号：TP393 **文献标志码**：A **文章编号**：0253-987X(2015)10-0084-06

A Detection Method of Injected Android Malicious Applications Using Function Calling Graphs

WANG Huan[1], LAI Huan[2], LI Guodong[3], TIAN Da[2], LIANG Bo[2]
(1. Apparel and Art Design College, Xi'an Polytechnic University, Xi'an 710048, China;
2. Xi'an Yanliang National Aviation Hi-Tech Industrial Base Management Committee, Xi'an 710048, China;
3. Network Center, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: The number of injected Android malicious applications is increasing, and the traditional detection method heavily relies on lots of extracted characteristics. A static analysis method based on function calling graph is proposed to detect injected Android malicious applications. The method can efficiently detect injected Android malicious applications only by analyzing the application's structure, and there is no need for known characteristics. The method constructs a function calling graph based on decompiled smali code, and sub-graphs will be further processed to tell whether the Android application is malicious or not. The period of analysis is much shorter than that of any other dynamic detection method. The detection method not only detects whether the Android application is malicious or not, but also has the ability to tell which part of the Android application contains malicious code. The approach is tested on 1 260 Android malicious applications and 1 000 Android normal applications, and the test results show that the approach is effective in detecting injected Android malicious applications. The detection rate of the method for the injected Android malicious applications is 95.94％ when the false positive rate is 8.90％.

收稿日期：2015-06-13。 作者简介：王欢(1980—)，女，讲师。 基金项目：陕西省科技统筹创新工程计划资助项目(2013KTCQ01-51)；国家自然科学基金资助项目(61103241)。
网络出版时间：2015-07-28 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20150728.1755.006.html

A comparison with the mainstream Android malware detection system Androguard shows that the detection performance of the proposed method has a remarkable promotion.

Keywords: Android; malware; static analysis; function calling graph

随着智能手机的快速发展，Android 操作系统成为当前最流行的移动终端平台。IDC 的统计数据显示，2014 年第 4 季度 Android 市场占有率达到 76.6%[1]。Android 系统的快速发展也伴随着 Android 恶意应用的出现，攻击者为了更好地传播 Android 恶意应用，会采用注入技术将恶意代码重打包到当下流行的 Android 应用程序中，并通过第三方电子市场、Android 应用程序分享论坛等途径进行传播。Zhou 的研究结果表明，第三方电子市场上 5% 至 13%的应用程序属于注入型 Android 恶意应用[2]。

Android 恶意代码静态检测研究方面，文献[3]基于 API 信息构建 Android 恶意应用的检测模型，取得了较好的检测效果。Wu 等提取 Android 应用程序中的权限、组件以及 Intent 等信息构建特征向量，并利用 k-均值算法建立恶意代码检测模型[4]。

Android 恶意代码动态检测研究方面，文献[5-6]分别实现了动态行为监控的沙盒系统，可在不同层次监控应用程序的运行行为。DroidScope 系统则进一步重新构建了 Android 系统与 Dalvik 虚拟机，实现了 Android 恶意代码的动态检测[7]。

本文针对注入型 Android 恶意应用提出了基于函数调用关系的恶意代码检测方法。为了保证应用程序功能正常执行，大部分注入型 Android 恶意应用通过注入的独立组件执行恶意行为，例如注入一个广播接收器用于监听开机启动事件，当手机重启后，该广播接收器便可被触发用于执行恶意行为。此种注入型 Android 恶意应用的特征使得恶意代码部分与正常应用代码之间在函数调用关系上联系较弱。

对于待检测的 Android 应用程序，首先进行反编译并在 smali 代码级别构建函数调用关系图，在此基础上将函数调用关系图进一步划分为函数调用子图，进一步基于子图中敏感 API 的调用次数计算子图威胁度，最后根据威胁度判定子图中是否包含恶意代码。

本文提出了针对注入型 Android 恶意应用的检测方法，所提方法基于静态分析，相对于动态分析系统，具有较高的代码覆盖率，同时也更加简单易行。针对 1 260 个 Android 恶意应用和 1 000 个正常应用样本的测试表明，所提方法针对注入型 Android 恶意应用具有很好的检测效果。

1　Android 应用程序基本结构

Android 应用程序以 APK 文件的形式发布，结构上主要包括 4 大组件，分别为：①Activity 组件提供与用户交互的界面，每个应用程序一般都存在一个主 Activity 作为应用程序启动的入口；②Service 与 Activity 不同的是，Service 组件主要运行在后台。Android 恶意应用为了防止被用户察觉，一般会通过 Service 在后台悄悄执行恶意行为；③Broadcast Receiver 组件接受来自应用程序内部其他组件或者其他应用程序发送的广播消息，因此可以用于监听系统的广播事件，例如网络状态变化、电池电量低、短信接收等；④Content Provider 组件提供数据操作的统一接口，可以通过 URI 声明数据访问地址，允许应用程序之间的数据共享。

2　案例分析

Geinimi 是 Android 平台上首个僵尸网络恶意应用，采用注入技术将其自身注入到流行的 Android 应用程序中，供 Android 用户下载，实现广泛传播的目的[8]。该恶意应用会收集被感染用户的隐私信息并可以接受远程控制服务器下发的命令，可对广大用户的个人隐私信息造成严重的威胁。以 MD5 值为 08E4A73F0F352C3ACCC03EA9D4E9467F 的恶意样本为例，从 AndroidManifest.xml 配置文件中可以发现该样本添加了一个名为 com.geinimi.AdServiceReceiver 的广播接收器，用于监听开机广播事件，并进一步触发恶意行为的执行。

图 1 为该样本的函数调用关系图，图中每个节点代表了一个方法，节点之间的边表示方法之间存在调用关系，空心节点表示调用的敏感 API，其他节点皆为非敏感 API。右上方的节点来自于应用程序中的正常代码部分，而左下方的节点则是来自于注入的恶意代码 Geinimi。从函数调用关系图可以看出，在该应用程序的正常代码部分，基本没有对敏感 API 的调用，而恶意代码部分则调用了大量敏感 API 用于实现恶意行为。另一方面，该样本的函数调用关系图明显地分为两个部分，同时两个部分之间没有明显的联系。通过对该样本的分析可以发

现，函数调用关系在一定程度上反映了 Android 应用程序内部各部分之间的关系，注入型 Android 恶意应用内部注入的恶意代码部分与正常代码部分之间存在明显的割裂性。

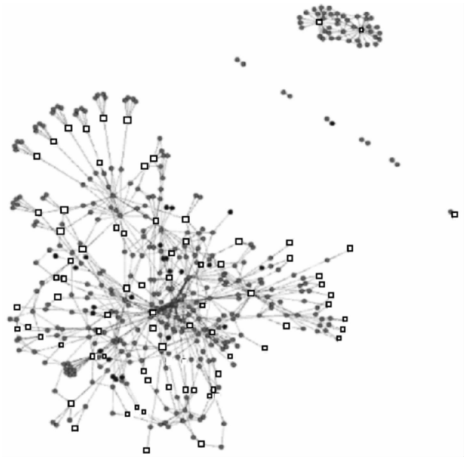

图 1 Geinimi 样本函数调用关系图

3 基于函数调用关系的检测方法

本文提出的基于函数调用关系的检测方法如图 2 所示，主要包含两个模块：函数调用关系图构建模块负责构建 Android 应用程序的函数调用关系图；恶意代码检测模块则负责识别函数调用关系图中包含恶意代码的部分。

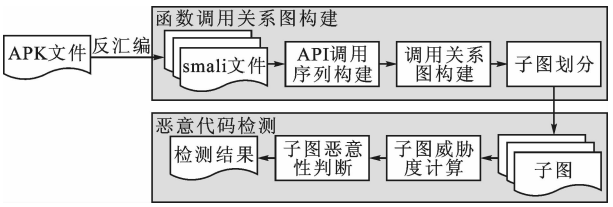

图 2 注入型 Android 恶意应用检测基本算法

3.1 函数调用关系图的构建

在函数调用关系图的构建过程中，主要包括 APK 反汇编、API 调用序列提取、函数调用关系图构建以及子图划分 4 个部分。

(1)APK 反汇编：在构建函数调用关系图之前，需要获得 APK 中调用的所有方法。为了获得精确的函数调用关系，首先利用 Android 应用程序逆向分析技术将待分析的 APK 文件经过反汇编得到细粒度的 smali 代码，并进一步基于 smali 代码构建函数调用关系。

(2)API 调用序列提取：在反汇编生成的 smali 代码基础上，从应用程序的每个入口点处搜索并回溯以构建 API 调用序列，函数的返回值以及参数可作为附加信息进行提取，进而构建信息更为丰富的 API 调用序列。

(3)函数调用关系图构建：Android 应用程序在开发过程中，主要以类和包的形式进行组织，开发者在开发过程中将具有一定关联的代码放入一个类中，进而组织到一个包下，因此类和包具有特定的语义信息。本文根据此语义信息，将前述提取的 API 调用序列基于类进行聚合，从而构建出函数调用关系图。

(4)子图划分：本文主要采用深度优先遍历算法在函数调用关系图的基础上进行子图划分，在对函数调用关系图进行遍历后，可以将调用的方法聚类到不同的子图中，同一个子图中的方法之间具有强相关性，而不同子图中的方法具有弱相关性。为了进一步识别子图中是否包含恶意代码，首先需要解决两个问题，一个是 Android 应用程序中存在的代码碎片问题，另一个是 Android 应用程序中广泛存在的广告模块问题。本文所指的代码碎片是指存在于应用程序中但没有实际功能的代码片段，其存在对后续的恶意代码识别具有干扰作用。为排除代码碎片的影响，本文提出了简单有效的方法用于识别 Android 应用程序中的代码碎片，其基本规则包括两方面：当子图中仅有一个类，且该类没有敏感 API 调用并且类中的方法数不超过 4 个时，该子图被标识为代码碎片；当子图中方法的方法体为空时，该子图被标识为代码碎片。

Android 应用程序开发者通常会在应用程序中添加广告模块，通过广告分成的方式获取收益。开发者通常可以在不修改应用程序整体代码的情况下利用广告软件开发包方便地集成广告模块，广告模块的实现机制导致广告模块在函数调用关系图上呈现独立的子图。另外，为了实现精确推送，部分广告模块会通过调用敏感 API 来收集用户的隐私信息，例如地理位置、浏览器搜索历史等。图 3 为包含广告模块应用程序的函数调用关系图，右上方的部分来自某广告模块，由图可见，该广告模块中同样包含了针对若干敏感 API 的调用。因此，广告模块与注入型恶意应用在函数调用关系图上具有相似的部分，在恶意代码检测过程中会导致误报。

为了区别广告模块和注入型 Android 恶意应用，需要对广告模块进行识别。通过分析发现，每个广告模块都有其对应的唯一包名，因此，可以基于广告模块的包名建立白名单，并依据此白名单对广告模块进行过滤，从而降低误报率。

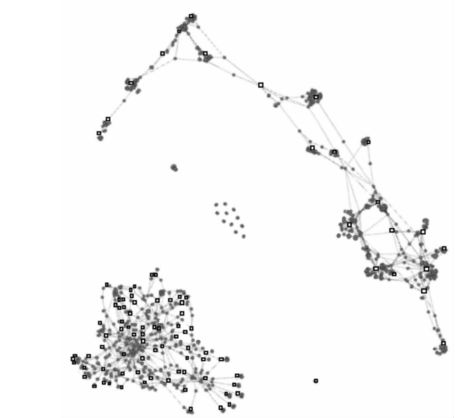

图 3　包含广告模块应用程序的方法调用图

3.2　恶意代码检测

在完成函数调用关系图构建与子图划分处理后，需要进一步判定 Android 应用程序中是否存在恶意代码。恶意代码的判定主要包括两个步骤：子图威胁度计算以及子图恶意代码判定。

(1)子图威胁度计算：Android 恶意代码通过调用敏感 API 来实现恶意行为，例如发送短信需要调用 sendTextMessage 函数，因此可以通过对敏感 API 的调用情况来计算子图的威胁度，本文首先通过对 1 000 个样本的统计分析得到了 Android 恶意代码中使用频率较高的 API 列表，表 1 显示了使用率最高的前 10 个 API 以及 API 的调用次数。每个敏感 API 会根据其潜在的威胁以及被恶意代码调用的次数赋予相应的威胁度值。对于每个子图，会进一步根据敏感 API 调用的次数计算子图的威胁度值，子图威胁度值为所有敏感 API 的威胁度值总和。

表 1　使用率最高的前 10 个系统 API

序号	函数名称	调用次数
1	ContentResolver. query	4 217
2	URL. openConnection	3 821
3	Runtime. exec	2 745
4	SmsManager. sendTextMessage	2 311
5	File. createNewFile	1 736
6	FileOutputStream. write	1 559
7	TelephonyManager. getDeviceId	1 468
8	Telephony. SMS_RECEIVED	1 011
9	BroadcastReceiver. onReceiveIntent	931
10	BroadcastReceiver. abortBroadcast	813

(2)子图恶意代码判定：在对 100 个注入型 Android 恶意应用程序的子图威胁度进行分析后发现，包含恶意代码的子图具有较高威胁度的同时其函数(所有函数)调用数目较少，而正常子图在具有较低威胁度的同时函数调用数目较多，因此可以进一步将子图威胁度值除以函数调用次数获得平均子图威胁度值。平均子图威胁度值越高，其包含恶意代码的可能性越大，为了确定恶意代码判定的阈值，对上述 100 个注入型恶意应用样本进行检测实验，结果如图 4 所示。当阈值超过 0.4 以后，恶意代码子图识别出现了漏报，且随着阈值的提高，漏报率逐渐上升；而当阈值低于 0.4 时，正常功能代码子图部分出现了误报，且随着阈值的降低，误报率快速上升。因此，本文将阈值设为 0.4，当平均子图威胁度值超过该阈值时，则判定该子图包含恶意代码。

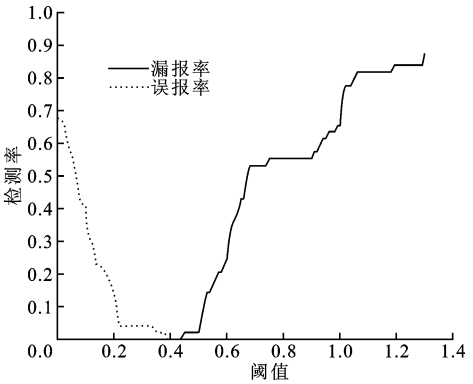

图 4　恶意代码子图漏报率和误报率随阈值变化曲线

4　实验结果与分析

本节对 Android Malware Genome Project[9] 中包含的 1 260 个恶意代码样本进行实验，测试本文方法对真实 Android 恶意代码的检测效果，同时与 Androguard[10] 进行对比，Androguard 主要是基于特征匹配进行恶意代码检测。实验过程中首先对应用程序样本进行反汇编得到 smali 代码，其次在 smali 代码的基础上提取 API 调用序列并构建函数调用关系图，同时进一步进行子图划分，最后对每部分子图进行威胁度计算，判断子图是否包含恶意代码。表 2 显示了对注入型 Android 恶意应用的最终检测结果，表 3 则为对非注入型 Android 恶意应用的检测结果。由实验结果可见，本文方法能够很好地检测注入型 Android 恶意应用，平均检测率达到 95.94%，远高于 Androguard 32.63% 的平均检测率；从对非注入型恶意应用检测效果来看，仍然可以检测出部分恶意应用。与基于特征的静态恶意代码检测系统不同的是，本文方法除了需要敏感 API 信息外不需要任何其他先验特征信息。Android 恶意

代码执行恶意行为一般需要调用系统 API 来实现，而这些 API 在一定程度上是稳定不变的，因此基于函数调用关系图的方法相比基于特征的恶意代码检测方法具有更好的稳定性，同时可以用于检测未知的恶意代码。

表 2 注入型 Android 恶意应用检测结果

恶意代码家族	样本数量	Androguard 检测率/%	本文方法检测率/%
BeanBot	8	0.00	100.00
Bgserv	9	0.00	100.00
Plankton	11	18.18	100.00
Zsone	12	100.00	100.00
DroidDream	16	93.75	100.00
jSMSHider	16	0.00	100.00
ADRD	22	59.09	100.00
YZHC	22	95.45	100.00
DroidKungFu1	34	0.00	97.06
DroidDreamLight	46	28.26	100.00
GoldDream	47	0.00	100.00
Pjapps	58	41.38	100.00
Geinimi	69	97.10	100.00
DroidKungFu4	96	0.00	72.92
AnserveBot	187	0.00	100.00
DroidKungFu3	309	0.00	99.35
其他	56	21.43	89.29

表 3 非注入型 Android 恶意应用检测结果

恶意代码家族	样本数量	Androguard 检测率/%	本文方法检测率/%
Asroot	8	0.00	50.00
RogueSPPush	9	100.00	55.56
SndApps	10	100.00	0.00
zHash	11	0.00	0.00
DroidKungFu2	30	0.00	60.00
KMin	52	78.86	21.15
BaseBridge	122	0.00	44.26

对于检测率低的恶意样本，我们将手动分析并解释本文方法所存在的局限性。

(1)Asroot：该恶意代码具有获取 root 权限的功能。该家族中的部分样本并没有采用注入技术将恶意代码注入到正常应用中，而是实现了一个具有恶意行为的完整 Android 应用程序，因此函数调用关系图之间没有明显的差异，这也导致了低检测率情况的发生。

(2)SndApps：该恶意代码样本实际上没有严重的安全威胁，其仅仅在正常应用中注入了大量广告模块并绑定了自己的广告 ID，以此来窃取其他开发者的广告分成利益。由于本文方法在子图划分过程中过滤了所有广告模块，因此对于 SndApps 的检测率为 0。

(3)zHash：该恶意代码样本通过添加垃圾代码实现变种目的，同时其自身也实现了部分对用户有用的功能，因此恶意代码与正常代码之间具有较强的相关性，从函数调用关系图来看，两个部分是一个整体，这也必然导致了基于函数调用关系检测方法失效。

进一步随机选取 1 000 个来自 Google Play 的正常 Android 应用程序作为数据集，测试了本文方法的误报率。表 4 显示了被检测出包含恶意代码的包名以及样本数量，一共有 89 个样本被标定为恶意代码，因此误报率为 8.90%。但是进一步分析发现，大多数被标定为包含恶意代码的包是来自于广告模块或者其他第三方模块。例如，com.smaato.SMOA 来自于名为 SMAATO 的广告模块；com.phonegap 来自第三方模块 PhoneGap，该模块可以为 Android 应用程序提供 Web 开发技术。因此，可以通过扩展白名单降低误报率。

表 4 正常 Android 应用程序检测结果

序号	检测为包含恶意代码的包名	样本数量
1	com.phonegap	20
2	org.acra	16
3	com.airpush.android	14
4	com.facebook.android	11
5	com.smaato.SOMA	10
6	com.mobfox.sdk	5
7	com.mobclix.android.sdk	4
8	com.Leadbolt	3
9	其他	6

5 结 论

本文提出了针对注入型 Android 恶意应用的检测方法，首先从待分析的 Android 应用程序中提取 API 调用序列，基于此调用序列进一步构造函数调用关系图。在函数调用关系图的基础上进行子图划分，通过计算平均子图威胁度值来判定是否包含恶意代码。实验结果表明，本文方法对于注入型 Android 恶意应用具有很好的检测效果。本文方法不

依赖于先验特征，可以用于检测已知和未知 Android 恶意应用。实验部分选取的样本较老，随着注入型 Android 恶意应用的技术发展，可能会出现新型的样本，因此在后续工作中，会继续对新发现的注入型 Android 恶意应用进行分析，进一步改进检测方法。

参考文献：

[1] IDC. Smartphone OS market share, Q4 2014 [EB/OL]. (2015-01-20) [2015-03-12]. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] ZHOU Wu, ZHOU Yajin, JIANG Xuxian, et al. Detecting repackaged smartphone applications in third-party Android marketplaces [C]// Proceedings of the Second ACM Conference on Data and Application Security and Privacy. New York, USA: ACM, 2012: 317-326.

[3] AAFER Y, DU Wenliang, YIN Heng. DroidAPIMiner: mining API-level features for robust malware detection in Android [C]// Proceedings of the 9th International Conference on Security and Privacy in Communication Networks. Berlin, Germany: Springer, 2013: 86-103.

[4] WU Dongjie, MAO Chinghao, WEI Teen, et al. Droidmat: Android malware detection through manifest and API calls tracing. [C]// Proceedings of the 7th Asia Joint Conference on Information Security. Piscataway, NJ, USA: IEEE, 2012: 62-69.

[5] GRACE M, ZHOU Yajin, ZHANG Qiang, et al. Riskranker: scalable and accurate zero-day Android malware detection [C]// Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. New York, USA: ACM, 2012: 281-294.

[6] ISOHARA T, TAKEMORI K, KUBOTA A. Kernel-based behavior analysis for Android malware detection [C]// Proceedings of the 7th International Conference on Computational Intelligence and Security. Piscataway, NJ, USA: IEEE, 2011: 1011-1015.

[7] YAN L K, YIN Heng. DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis [C]// Proceedings of the 21st USENIX Conference on Security Symposium. Berkeley, CA, USA: USENIX, 2012: 29.

[8] PIETERSE H, OLIVIER M S. Android botnets on the rise: trends and characteristics [C]// Proceedings of the Conference on Information Security for South Africa. Piscataway, NJ, USA: IEEE, 2012: 1-5.

[9] JIANG Xuexian, ZHOU Yajin. Android malware genome project [EB/OL]. (2012-08-11) [2015-03-12]. http://www.malgenomeproject.org.

[10] DESNOS A. Androguard: reverse engineering, malware and goodware analysis of Android applications and more [EB/OL]. (2013-11-21) [2015-03-12]. http://code.google.com/p/androguard.

（编辑　武红江）

（上接第 83 页）

[4] SHAO Xing, WANG Ruchuan, XU Hai, et al. Genetic algorithm based coding aware routing for wireless mesh networks [J]. Advances in Information Sciences and Service Sciences, 2012, 4(5): 752-763.

[5] CHOID J, BANG H, LEE C. Network coding-aware multicast optimization in multi-hop wireless networks [C]// Proceedings of the 2014 IEEE International Conference on Consumer Electronics. Piscataway, NJ, USA: IEEE, 2014: 466-467.

[6] DE C, ARACHCHI H K, FEMANDO A, et al. Content and network-aware multicast over wireless networks [C]// Proceedings of the 2014 10th IEEE International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness. Piscataway, NJ, USA: IEEE, 2014: 122-128.

[7] BENFATTOUM Y, MARTIN S, AGHA K A. QoS for real-time reliable multicasting in wireless multi-hop networks using a generation-based network coding [J]. Computer Networks, 2013, 57(6): 1488-1502.

[8] 黄传河，杨文忠，王博，等. 无线 Mesh 网中编码感知组播路由协议 CAMR [J]. 计算机研究与发展，2011，48(6)：1000-1009.
HUANG Chuanhe, YANG Wenzhong, WANG Bo, et al. CAMR: network coding-aware muticast routing protocol in wireless mesh networks [J]. Journal of Computer Research and Development, 2013, 48(6): 1000-1009.

[9] ROYER E M. Multicast operation of the Ad-hoc on-demand distance vector protocol [C]// Proceedings of the Annual International Conference on Mobile Computing and Networking. New York, USA: ACM, 1999: 207-218.

（编辑　武红江）