一种基于子图搜索的特征定位方法

付 焜 吴毅坚 彭 鑫 赵文耘
(复旦大学软件学院 上海 201203) (上海市数据科学重点实验室(复旦大学) 上海 201203)

摘 要 确定源代码中与给定需求特征相关的程序元素的过程称为特征定位。但现在大部分的特征定位方法仍是以特征相关的描述和代码结构信息为输入,以特征相关的代码元素为输出。这样的结果缺乏代码元素间的关系信息,难以让开发者快速理解相关特征的程序结构。针对这个问题,提出一种基于子图搜索的特征定位方法。该方法能找出与特征相关的代码元素,并以依赖调用图的形式将结果展示出来,让开发者快速了解代码结构。根据该方法实现了相应工具并经过实验验证了方法的有效性。该方法的平均准确率为40.41%,平均召回率为50.28%。

关键词 特征定位,程序理解,代码结构,调用依赖
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.012

Feature Location Method Based on Sub-graph Searching

FU Kun WU Yi-jian PENG Xin ZHAO Wen-yun
(School of Software, Fudan University, Shanghai 201203, China)
(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract The process to identify relevant program elements according to a given feature is called feature location. However, existing feature location methods, mainly based on feature description and source code structure, only produce source code elements as the result, which is usually lack of structural information and makes it difficult for developers to understand the code structure quickly. To solve this problem, a feature location method based on sub-graph search was proposed. The method finds out code elements related to the feature and the results can be displayed in a call graph. The method is implemented as a tool and tested for its performance. The average precision is 40.41% and the average recall is 50.28%.

Keywords Feature location, Program understanding, Code structure, Call dependence

1 引言

为了进行错误修复、增强原有功能等软件维护任务,开发者需要了解相关的功能在代码中是如何分布以及如何实现的。在软件工程中,确定与特定功能相关的实现代码的过程称为特征定位[1-2]。特征定位是软件维护和评价过程中最为重要和普遍的活动之一。如果没有在开始阶段将与现有任务相关的代码在程序中定位出来,那么软件维护工作将无法完成[2]。

传统的特征定位技术已经发展得比较成熟,例如信息索引技术[3]、静态分析技术[4]、动态分析技术[5]以及多种技术相结合的方法[9]。但是,传统的特征定位技术大多都只将与特征相关的代码元素返回给用户,这些元素彼此之间没有明确的关系,开发者在获取到这样的结果后,往往无法直接看出这些元素是如何相互合作完成特定功能的,因此开发者仍然需要耗费一定的精力来整合所得的结果[11]。

文中提出一种结合了基于文本的方法和基于静态分析的特征定位方法。对于给定的软件项目,用户输入特征关键字后,所提方法能够返回项目方法依赖图的一个子图。这样的子图不仅包含特征相关的方法元素,还包含方法元素间的依赖关系,让用户在查看代码细节之前就能够在一定程度上了解代码的实现过程,有助于用户尽快了解代码的结构。基于该方法,实现了可以处理Java项目的特征定位工具,并通过实验验证了该方法的有效性。

2 动机和试探性研究

尽管特征定位方法近年来得到了广泛的研究,然而现有的方法都无法高效地返回代码元素之间的结构信息。例如,Kevic等[6]将特征定位的结果以顺序图的形式返回给用户,能够帮助用户更快地了解方法之间的调用关系,但其源代码定位部分相对简单,没有很好地利用代码元素之间的结构关系;同时,由于代码之间的依赖关系相对复杂,仅仅用顺序图

到稿日期:2015-11-30 返修日期:2016-03-02 本文受国家自然科学基金(61370079),国家高技术研究发展计划(863)(2013AA01A605)资助。

付 焜(1989-),男,硕士生,主要研究领域为软件维护与演化、特征定位,E-mail:luminosite@163.com;吴毅坚 博士,副教授,主要研究领域为软件体系结构、软件复用和产品线、工业环境中的软件演化;彭 鑫 男,博士,副教授,主要研究领域为软件维护与演化、软件产品线、自适应软件、移动计算与云计算;赵文耘 男,教授,主要研究领域为软件工程、软件开发工具及其环境、企业应用集成(EAI)。

无法充分展示代码间复杂的关系。Wang 等[7]提出了一种交互式的、基于多刻面的特征定位方法，该方法虽然能返回代码元素间部分的依赖关系，但仍然难以让用户尽快了解相关代码的结构，且需要用户与工具进行多次交互。

相比较而言，基于子图搜索的特征定位方法根据用户输入的关键字自动搜索相关代码元素，并将特征定位结果以一个方法依赖图的形式呈现给用户，使得用户对相关代码及其结构有最直观的认识。然而，就特征定位而言，要求源代码本身具有良好的“聚集性”。这种聚集性是指代码元素在方法依赖图上能聚集在一起，即对于一个特征，在源代码的方法依赖图中能够找到一个子图，其包含所有与该特征相关的代码元素，并且包含较少的与特征不相关的代码元素。为了验证特征对应的代码元素具有良好的聚集性，对文献[2]中公布的一个特征定位标准结果集进行了试探性研究。此数据集中包含了 6 个开源项目，每个开源项目包含若干特征描述，每个特征描述包含了其对应的方法级代码元素。我们选取了 JEdit 项目，它共包含 150 个特征及其描述。这 150 个特征所对应的代码元素的数量差异较大，最少的对应 1 个代码元素，最多的对应了 22 个代码元素，而绝大部分特征对应了 2～7 个代码元素。由此我们认为 2～7 个代码元素为特征对应代码元素个数的一个普遍范围，因此后续调研过程中在对应 2～7 个代码元素的特征中选取了 58 个特征进一步分析代码的聚集性。首先通过静态分析，在方法层次上还原了 JEdit 项目中代码元素之间的依赖关系图 $G(V_G, E_G)$，其中 V_G 表示方法集合，有向边集合 E_G 表示依赖关系集合；然后对于每一个特征，尝试在 JEdit 的关系依赖图 G 中搜索一个子图 $GS(V_S, E_S)$，使得如下 3 个公式同时成立：

$$V_S \subseteq V_G \tag{1}$$

$$E_S \subset E_G \tag{2}$$

$$V_{Feature} \subseteq V_S \tag{3}$$

其中，$V_{Feature}$ 代表特征对应的代码元素，也是依赖图 G 上的顶点集 V_G 的一个子集。为了验证这些代码元素的聚集性，我们期望找到的 G_S 是尽量不包含 $V_{Feature}$ 之外的代码元素的图。为了搜索 G_S，引入斯坦纳树[8]，它是使用最短连接路将图中特定点集合连接起来的一个子图；当图中边的权重相同时，它也是能连接特定点集合并保证引入最少的特定点集合之外的点的子图。因此，以 $V_{Feature}$ 作为特定点集合在图 G 上搜索到的斯坦纳树就满足 G_S 的限制条件。在进行调研的过程中，使用搜索斯坦纳树的搜索算法[8]对每一个特征对应的代码元素集合进行斯坦纳树搜索。搜索的结果如表 1 所列。

表 1　特征对应元素聚集性调研结果

特征数量	特征对应元素数量	斯坦纳树平均节点数	增长比例
18	2	3.33	1.67
8	3	5.50	1.83
11	4	7.82	1.96
7	5	8.00	1.60
7	6	9.57	1.60
2	7	13.50	1.93
平均值	3.68	6.40	1.74

表 1 中总共包含 53 个特征的调研结果，按照特征元素数量分为 6 类，列出了每类特征的数量、对应代码元素的数量、找到的斯坦纳树的平均节点数，以及增长比例即平均节点数和代码元素数量的比值，该比值是子图 G_S 中的代码元素数量与特征对应的代码数量的比例，比例越小，说明特征对应元素的聚集性越好。由表 1 可以看出，平均的增长比例小于 2，说明大部分代码元素的聚集性较好。调研中还有 5 个结果没有找到相应的斯坦纳树。经过人工检查，其中 2 个特征对应的代码元素中有个别代码元素与其他元素的依赖距离较远，导致搜索复杂度剧增而没有找到结果，而另外 3 个特征则是因为 JEdit 中使用了图形框架，丢失了一些依赖关系，导致元素无法关联在一起。这 5 个特征关联的代码除了个别元素外，其余代码元素的聚集度较好。以上就是以一个 Java 项目作为代表对特征对应代码元素聚集性的调研。从调研结果来看，特征对应的代码的聚集性比较好，可以使用本文方法进行特征定位。

3　方法步骤

方法的流程如图 1 所示。总体上分为两步，第一步是在源代码中搜索候选结果；第二步将根据源代码的依赖关系图和候选结果搜索出一个依赖关系图的子图，该子图被称为目标子图，是最后返回给用户的结果。

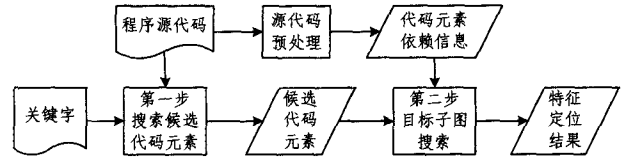


图 1　方法概览

3.1　候选结果的搜索和相似度计算

方法的第一部分是先使用基于文本的检索技术，从项目的源文件中检索与用户输入的关键字匹配的方法作为候选结果，然后计算每个方法与关键字的匹配程度。

候选结果的具体检索方法如下：对源代码中的每个方法代码段，检验是否出现了用户输入的关键字，如果出现任何用户输入的关键字，便把该方法元素列为候选结果。在方法实现的过程中，考虑到文本检索的效率问题，采用 Lucene 搜索引擎实现相关代码元素的检索，因此需要对项目数据实现导入 Lucene 引擎，构建索引。而在具体检索的过程中，将关键字作为 Lucene 搜索引擎的输入，检索出包含关键字的所有结果作为候选结果。

为了下一步的子图搜索，搜索到候选结果后，还要计算出每个候选结果与用户输入关键字之间的相似度。使用候选结果代码中与输入关键字相同的词语的个数作为候选结果的相似度，具体的计算公式如下：

$$Sim(Q,m) = |\{w | w \in Q \wedge w \in m\}| \tag{4}$$

其中，Q 为用户输入的关键字的集合，m 为方法的文本段中所包含的词语的集合。

3.2　子图搜索过程

在检索到候选结果后，就可以根据候选结果和源代码的依赖结构进行目标子图的搜索。但为了实现子图搜索，对项目的源代码数据进行了一些预处理，之后再进行具体的目标子图的搜索。

3.2.1 对项目源代码的预处理

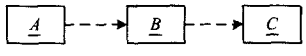
为了搜索目标子图，需要获取到能将代码中任意两方法元素间最短的、通过调用关系连接起来的路径，即关联路径，其具体定义将在后文详述。为了获取关联路径的数据，首先要分析出源代码中方法元素间的直接调用关系，然后利用直接的依赖关系构建任意两个方法元素间的关联路径。因此整个预处理过程可以分为两步：1)对源代码进行分析，获取方法元素间的直接依赖关系数据；2)利用弗洛伊德算法找出任意两个方法元素间的关联路径，并将结果记录下来。

(1)获取项目方法元素间的直接依赖关系

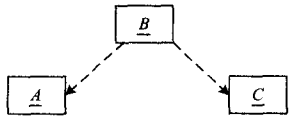
为了获取源代码中方法元素之间的依赖关系，需要对源代码进行静态分析。目前所提方法主要针对 Java 项目，所以选取了 Eclipse 项目下的 JDT 工具对项目进行分析。JDT 是 Eclipse 下对源代码进行分析的工具，调用 JDT 的相应接口就可以分析源代码，从而获取到源代码中方法元素之间的直接调用信息。

(2)任意两元素间依赖关系的计算

获取到源代码方法元素之间的直接调用关系后，需要进一步获取到任意两方法元素间的关联路径。关联路径为两方法元素间最短的、通过调用关系连接起来的路径。关联路径并不考虑路径中依赖关系的方向。如图 2 所示，图 2(a)为一条典型的由 A 方法到 C 方法的调用链，在没有其他从 A 到 C 的调用关系的情况下，这条调用链即为 A,C 方法之间的关联路径；而图 2(b)中，由 B 到 A 和由 B 到 C 的依赖关系同样能够组成 A,C 之间的关联路径。



(a)依赖方向一致的关联路径



(b)依赖方向不一致的关联路径

图 2 关联路径示意图

为了计算出任意两方法元素之间的关联路径，可以将该问题转化为求图上任意两点间的最短路径问题，而由于关联路径的定义中不考虑依赖关系的方向，因此可以将该问题具体转化为无向图上任意两点间的最短路问题。在实现过程中，可以先通过方法之间的调用关系数据构建整个项目的方法依赖图，在图上添加依赖关系的逆向关系将有向图转换为无向图。在此之后，使用弗洛伊德算法求解出图上任意两点之间的最短距离，同时在求解过程中记录下最短路径。该路径即为任意两方法元素之间的关联路径。最后将图的最短路径的数据存储到数据库中。在方法实现时，为了加快访问速度，我们会在方法初始化时将数据库中的数据加载到内存中，以便每次特征定位时能快速访问这些数据。

3.2.2 目标子图搜索

目标子图的具体定义如下：对于用户输入的关键字集合 Q，目标子图 $G_{target}(V_t,E_t)$ 是整个项目依赖图的一个子图，并且是使得子图的评价值 $Gain(G_s,Q,\rho)$ 最大的一个子图。对于给定的子图 $G_s(V,E)$ 和关键字集合 Q，以及给定的参数 ρ，子图评价值 $Gain(G_s,Q,\rho)$ 的定义如下：

$$Gain(G,Q,\rho)=\frac{\sum_{v_i\in V} Sim(Q,v_i)}{|Q|+\rho|E|} \tag{5}$$

子图中的点 v_i 对应源代码中的方法元素，所以其中 $Sim(Q,v_i)$ 为 3.1 节中定义的方法元素与输入的关键字之间的相似度；$|E|$ 为子图中边的总长度；ρ 为人工设置的参数，用来调节子图大小对评价值的限制作用。子图评价值的定义参考了文献[13]中对子图的估价函数的定义，该评价值可以保证目标子图是包含更多候选方法元素而同时尽量不包含非候选元素的子图，从而保证我们能够获得较好的特征定位结果。

由于搜索使评价函数达到最优的子图是一个组合问题，搜索的开销较大；而且在子图搜索的实际实现过程中，整个程序的方法依赖图数据量极大，实验中使用的 JEdit 项目总共包含了 6 千多个方法，因此若直接在依赖图上进行搜索，在算法性能上将无法接受。为了解决图搜索的效率问题，设计了一种基于普里姆算法的贪心算法来获取次优子图。而且根据实验，该算法获取到的准确率和召回率结果都比较好。获取次优解的算法如下：

首先在搜索到的候选结果集中，选取相似度最大的方法元素放到结果子图中，如果有多个相似度最大的方法元素，则随机选取一个。然后开始一个迭代的图扩展过程，在每一个迭代的过程中，都会遍历一次未被加入到子图中的候选元素。之后检查新元素到现有子图中所有元素的关联路径，取得最短的一条路径，以该路径和候选元素作为一个小的子图计算一个评价值。计算所有元素产生的新的子图评价值，然后取得最大的评价值对应的元素，并尝试着将该元素加入到现有的子图中。如果新子图的评价值比之前的评价值高，则继续进行迭代过程；否则不将新元素加入子图，并结束迭代过程。

正常的迭代过程会因为过多非相关的代码元素加入到子图中，使得子图的评价值下降而自动终止。但不能排除方法不能自动终止，或是方法自动终止时子图过大的情况。过大的子图会引入过多的不相关元素，同时也不利于用户理解。因此，在方法中添加一个子图大小的上限值 m。在迭代过程中，若候选元素数量到达上限，停止迭代。实验时，m 被设置为 16。迭代过程终止后，得到的子图即为最后返回给用户的特征定位结果。结果中包含了代码的结构信息，可以用可视化图形的方式将结果展示给用户。最终结果的展示效果如图 3 所示，具体展示形式和效果将在第 4 节介绍。

4 实验评估

为评估方法的有效性，基于该方法实现了处理 Java 项目的特征定位工具。从文献[2]中公布的 JEdit 项目的标准结果集中选取了 8 个特征进行实验。但标准结果集提供的特征描述较长，我们从描述中人工选取关键字作为输入，将结果与标准结果比对，从结果的准确率、召回率和 F-Measure[17] 值 3 方面进行评估。

4.1 实验结果

表 2 列出针对 8 个特征的实验结果。实验时式(5)中的参数 ρ 设置为 0.6，目标子图大小的上限值 m 设置为 16。表 2 第一列为实验时使用的关键字，第二列为实验结果的准确

率，第三列为实验结果的召回率，最后一列为 F-Measure 值。F-Measure 是一个综合了准确率(P)和召回率(R)的考察指标，按照下式计算：$F_\beta=(\beta^2+1)PR/(\beta^2P+R)$，其中 β 是参数，本次实验中取值为 1。

表 2　方法效果实验结果表/%

关键字	准确率	召回率	F-Measure
Hightlight, hypersearch	66.67	85.71	75.00
Gutter, select, text	33.33	18.75	24.00
Syntax style	37.50	66.67	48.00
Hash, save	6.25	20.00	9.25
Shortcut, table, filter	27.27	25.00	26.09
Indent, whitespace	27.27	75.00	40.00
select, status, text, counter	25.00	44.44	32.00
Fullscreen mode	100.00	66.67	80.00
平均值	40.41	50.28	41.83

从表 2 中可见，8 个实验的平均准确率为 40.41%，平均召回率为 50.28%，平均 F-Measure 为 41.83%，总体上在特征定位领域中该方法的准确性是可接受的。在特征定位过程中，为了完成与特征相关的维护任务，开发者更希望找全相关的代码元素，因此召回率更为重要，而该方法的召回率也是可以接受的。

但在实验中发现，以“hash”和“save”为关键字进行查询的准确率和召回率较低。再次查看该特征的简短描述：“MD5Sum a buffer and file for ‘file not saved’ warning check”。实验时使用“hash”概括了具体的 MD5 算法，所以我们尝试着直接使用“MD5”和“save”关键字进行搜索，结果得到的准确率为 23.53%，召回率为 80%，F-Measure 为 36.36%，各指标都得到了很大的提升。可见本文方法在一定程度上依赖于一开始输入的关键字。而事实上，一般情况下从特征描述中抽取关键字并不困难。从上面的特征描述例子和表 2 展示的总体较好的定位结果可以看出这一点。

4.2　实例分析

图 3 是我们工具特征定位的一个结果实例。其特征描述为“A fullscreen mode for JEdit would be very nice. Especially on netbooks with limited screen-size it is useful, to get rid of the titlebar and window-borders.”，输入的关键字为 Fullscreen mode。图上节点为方法元素，标有方法的完整名称，包括包结构、类名和方法签名。节点间的有向边代表方法元素间的调用关系。图中字体加粗的节点代表最初搜索到的候选元素，字体为斜体的节点为非候选结果。该案例的召回率为 66.67%，图 3 中包含了 6 个正确结果中的 4 个，即 4 个候选方法元素。

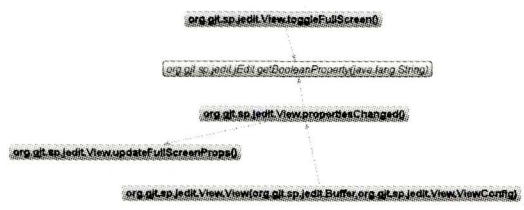


图 3　用例可视化结果

从图 3 中可以看到，View 方法构造窗口时会直接或间接地调用其他两个候选方法，并间接调用非候选方法 getBooleanProperty，而另一个候选方法 toggleFullScreen 也调用了该方法。开发者可以结合图中的方法名和调用结构猜测到，View 的构造函数在图形构件初始化时调用 propertiesChanged 和 updateFullScreenProps 以及 getBooleanProperty 几个方法来获取全屏属性的数据，并按照获取的数据将相应的界面显示出来。而另外的 toggleFullScreen 方法也调用了 getBooleanProperty。可见，toggleFullScreen 是用来实现图形构件初始化后再次更改显示方式的方法。这种猜测在我们看过具体代码的实现细节后进一步得到印证。可见，本文方法的结果能够有效地为用户提供结构信息，让用户在查看代码的实现细节之前就能对代码有整体上的了解，这有助于用户理解相应特征的实现过程。与本文方法相比，传统的定位方法无法返回代码的整体结构信息，无法直接向用户展现相关代码的结构。文献[4]中的方法也只能返回代码局部的调用顺序图，难以结合全局结构猜测代码的功能。而我们之前提出的特征定位方法返回的结果为方法调用链[3]，包含的结构信息很少，而且准确率和召回率较低，难以辅助用户理解特征的实现过程。

结束语　针对已有的大部分特征定位方法只返回与特征相关的代码元素列表的现状，文中提出了一种基于子图搜索的特征定位方法。该方法结合了基于语义信息和基于静态分析的特征定位方法，并返回带有结构信息的结果。我们在实现了新的基于方法调用子图搜索的特征定位方法的基础上，实现了一个工具并进行了实验，从而证明了此方法的有效性。最后对一个实际的特征定位搜索案例进行了分析，说明了所提方法的特征定位结果辅助使用者理解代码结构的作用。

参 考 文 献

[1] BIGGERSTAFF T J, MITBANDER B G, WEBSTER D E. Program understanding and the concept assignment problem[C]// Communications of the ACM. 1994: 482-498.

[2] DIT B, REVELLE M, GETHERS M, et al. Feature location in source code: a taxonomy and survey[J]. Journal of Software Maintenance & Evolution Research & Practice, 2012, 25(1): 53-95.

[3] MARCUS A, MALETIC J I. Recovering Documentation to-Source-Code Traceability Links using Latent Semantic Indexing [C]// ICSE. 2003: 125-135.

[4] TRIFU M. Using Dataflow Information for Concern Identification in Object-oriented Software Systems[C]// European Conference on Software Maintenance & Reengineering. 2008: 193-202.

[5] WONG W E, HORGAN J R, GOKHALE S S, et al. Locating Program Features using Execution Slices[C]// Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology, 1999 (ASSET'99). IEEE, 1999: 194.

[6] KOŠNAR K, VOÁSEK V, KULICH M, et al. CoMoGen: An approach to locate relevant task context by combining search and navigation[C] // IEEE International Conference on Software Maintenance & Evolution. IEEE, 2014: 61-70.

[7] ZHAO W. Improving feature location practice with multi-faceted interactive exploration[C]// 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 762-771.

(下转第 81 页)

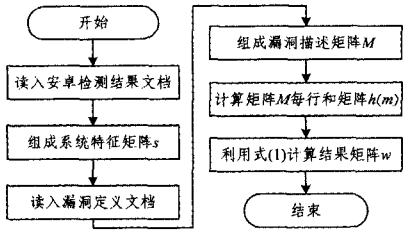


图 3　漏洞评估过程

4　实验与结果

实验主要考查精度和性能两方面,以客观反映系统对漏洞的识别率和性能。

测试环境:将控制台安装在一台装有 Windows 操作系统的台式电脑上。将代理端安装在一台中兴 N798(电信版)安卓手机上,此手机 CPU 为双核 1.0GHz,RAM 为 512MB,操作系统为 Android OS 4.0。

4.1　评估精度测试

在上述测试环境中进行测试。由于安卓系统版本较高,安全性增加,系统共检测到 3 个漏洞,如表 1 所列。

表 1　漏洞检测系统检测到的 3 个漏洞信息

CVE ID	漏洞描述
CVE-2012-2808	Android 4.0.4 DNS Poisoning
CVE-2013-6770	Android 4.3 Superuser Root Privilege Escalation
CVE-2013-6774	Android 4.2.x SuperuserUnsanitized Environment

通过 CVE 数据库得知安卓官网已提供对应漏洞的修补方法。将安卓手机系统升级后,重新用此系统对手机进行漏洞检测评估,3 个漏洞的值均由真转为假。

4.2　评估性能测试

为了测试原型的可伸缩性和内存调用。实验从 5 个 OVAL定义开始,每次增加 5 个 OVAL 定义,直到 OVAL 定义增加至 100 个。在每次实验中查看安卓手机的 CPU 利用率和内存占用情况,CPU 利用率和内存占用情况分别如图 4 和图 5 所示。

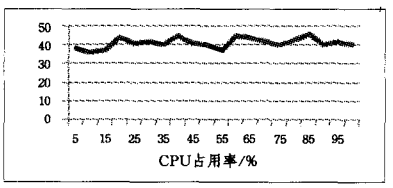


图 4　CPU 占用率统计结果

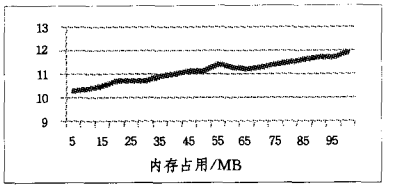


图 5　内存占用统计结果

结束语　本文提出一种基于 OVAL 的安卓漏洞检测评估系统,实验测试结果充分表明该系统具有以下特点:漏洞检测评估精度高;基于 OVAL,评估结果精度高并且能与其他安全产品共享利用;采用 C/S 的架构降低了对安卓手机性能的影响。

参 考 文 献

[1] ENCK W,ONGTANG M,MCDANIEL P. Understanding Android Security[J]. IEEE Security & Privacy Magazine,2009,7(1):50-57.

[2] SHABTAI A,FLEDEL Y,KANONOV U,et al. Google Android:A Comprehensive Security Assessment[J]. IEEE Security & Privacy,2010,8(2):35-44.

[3] BARTEL A,KLEIN J,TRAON Y L,et al. Automatically securing permission-based software by reducing the attack surface:an application to Android[C]// Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM,2012:274-277.

[4] HANNA S,HUANG L,WU E,et al. Juxtapp:A Scalable System for Detecting Code Reuse among Android Applications[M]// Detection of Intrusions and Malware,and Vulnerability Assessment. Springer Berlin Heidelberg,2013:62-81.

[5] The MITRE Corporation. OVAL[EB/OL]. (2015-07-09)[2015-11-15]. http://oval.mitre.org.

[6] The MITRE Corporation. CVE[EB/OL]. (2015-07-24)[2015-11-15]. http://cve.mitre.org.

[7] Internet Security SystemsTM. Vulnerability assessment[EB/OL]. (2015-07-26)[2015-11-15]. http://www.iss.net/find_products/vulnerability-assessment.php.

[8] WANG X D,GAO L,ZHANG L. Design and implementation of OVAL-compatible VAS on multi-platform[J]. Computer Engineering and Applications,2009,45(36):82-85. (in Chinese)
王旭冬,高岭,张林. 兼容 OVAL 的多平台 VAS 设计与实现[J]. 计算机工程与应用,2009,45(36):82-85.

(上接第 59 页)

[8] BYRKA ,JAROSŁAW,GRANDONI F,et al. An Improved LP-based Approximation for Steiner Tree[C]// Proceedings of the Forty-second ACM Symposium on Theory of Computing. ACM,2010:583-592.

[9] ZHAO W,ZHANG L,LIU Y,et al. SNIAFL:Towards a Static Non-Interactive Approach to Feature Location[C]// International Conference on Software Enginee-ring. IEEE Computer Society,2004:293-303.

[10] EISENBARTH T,KOSCHKE R,SIMON D. Locating Features in Source Code[J]. IEEE Transactions on Software Engineering,2003,29(3):210-224.

[11] FU K,QIAN W Y,PENG X,et al. Feature Location Method Based on Call Chain Analysis[J]. Computer Science,2014,41(11):36-39. (in Chinese)
付焜,钱文亿,彭鑫,等. 一种基于调用链分析的特征定位方法[J]. 计算机科学,2014,41(11):36-39.

[12] BYRKA,JAROSŁAW,GRANDONI F,et al. An Improved LP-based Approximation for Steiner Tree[C]// Proceedings of the Forty-second ACM Symposium on Theory of Computing. ACM,2010:583-592.