结合项约束的闭合模式挖掘研究*)

王新宇　唐世渭
(北京大学信息科学中心　北京100871)

摘　要　事务数据库中频繁模式的挖掘研究作为关联规则等许多数据挖掘问题的核心工作，已经研究了许多年。然而，频繁模式挖掘算法经常产生大量的模式和规则，不但降低了算法的执行效率，同时也使用户从频繁模式产生有用的规则变得很困难。针对这个问题，最近的研究主要集中于两点，一种方法是允许用户附加约束来引导挖掘的过程，通过把约束条件下推到挖掘的底层来缩小模式搜索的空间，提高性能；另一种方法是仅挖掘闭合模式，只产生大于其超集支持度的频繁模式。两种方式都可以大量缩小结果集合的大小，使结果集合更容易被用户理解和使用。那么，把这两种方式相结合，挖掘满足用户约束的闭合频繁模式，理论上来说应该更为高效，更方便理解和使用。基于以上的考虑，做了一些细致的研究，把用户约束分类，并主要讨论了结合项约束的闭合模式生成问题。

关键词　频繁闭合模式，约束，关联规则，数据挖掘，算法

Frequent Closed Pattern Mining with Item Constraints

WANG Xin-Yu　TANG Shi-Wei
(School of Electronics Engineering & Computer Science, Peking University, Beijing, 100871)

Abstract　Mining frequent patterns in transaction databases, as an essential role in many data mining tasks such as the association rule mining, has been widely studied for many years. However, frequent pattern mining often generates a very large number of patterns and rules, which reduces not only the efficiency but also the effectiveness of mining. Recent work focuses mainly on two aspects. One is to mine frequent closed patterns; the other is to mine frequent patterns with user's constraints. These two points can both reduce the number of result patterns; make the users easier to understand. Therefore theoretical, if we combine these two points to mine frequent closed patterns with user's constraints, the result set should be more easier to understand. Based on the above, we make some careful research, classify the constraints, and mainly focus on the problem of mining frequent closed patterns with item constraints.

Keywords　Frequent closed pattern, Constraint, Association rule, Data mining, Algorithm

1　介绍

自 Agrawal 等人[1]于1993年提出频繁模式挖掘问题以来，已经提出了许多行之有效的技术[2~4]来进行频繁模式挖掘，根据挖掘的目标和应用领域的不同，这些挖掘方法可以分为两大类：产生候选集/候选模式的方法和不产生候选集/候选模式的方法。前一种方法以算法 Apriori[2]为代表，而后一种以算法 FP-growth[4](Frequent Pattern growth)为代表。FP-growth 算法主要用于长模式与富模式的挖掘。

然而，频繁模式挖掘经常产生大量的模式和规则，不但降低了算法的执行效率，同时也使用户从频繁模式产生有用的规则变得很困难。针对这个问题，最近的研究主要集中于两点：一种方法是允许用户通过加以约束来引导挖掘的过程[5,6]，通过把约束条件下推到挖掘的底层来缩小模式搜索的空间，提高性能；另一种方法是仅挖掘闭合模式(closed pattern)[7~11]，只产生与其超集支持度不同的频繁模式。闭合模式的挖掘在保持信息的完整性的前提下，很大程度地减少结果集的大小，从这个精简的结果集中，可以很直接地产生所有的频繁模式。

两种方式都可以大量减少结果集合的大小，使结果集合更容易被用户理解和使用。那么，把这两种方式相结合，挖掘满足用户约束的闭合频繁模式，理论上来说应该更为高效，更容易被用户理解和使用。基于以上的考虑，我们做了一些细致的研究，把用户约束分类，主要讨论了基于项约束的闭合模式生成问题。

2　问题定义和相关工作

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是项的集合。设任务相关的数据是多个事务的集合，简称为事务集，并标记为 D。D 中每个事务 T 是项的集合，使得 $T\subseteq I$。每一个事务有一个标识符，称做 TID。设 A 是 I 中不同项的集合，称为项集或者模式，事务 T 包含 A 当且仅当 $A\subseteq T$。包含 k 个项的模式称为 k-模式。模式 A 的支持度计数是指事务集中包含该模式的事务数，记为 $A.count$，其支持度 $sup(A)$ 定义为 $A.count/|D|$，其中 $|D|$ 代表事务集中所含事务的总数。给定一个最小支持度 min_sup，当事务集保持不变时，模式 A 满足最小支持度 min_sup 等价于它的支持度计数 $A.count$ 大于或等于 min_sup 与事务总数的乘积。最小支持度 min_sup 与事务总数的乘积记为最小支持度计数 $min\_supcount$。如果模式 A 满足 $A.count\geqslant min\_supcount$，则称 A 为频繁模式(frequent pattern)。频繁 k-模式的集合通常记作 F_k。对于 I 中单个的项，如果它在 D 的事务中出现次数 $\geqslant min\_supcount$，则称该项为频繁项。

2.1　频繁闭合模式

定义1(频繁闭合模式)　如果一个模式 Y 是频繁的，并且它的所有的超集 Y' 的支持度都小于 Y 的支持度，那么 Y 称为闭合频繁模式。

*)本文的研究得到国家重点基础研究发展规划(973)项目(No.G1999032705)资助。王新宇　硕士研究生，主要研究方向为数据仓库、数据挖掘。唐世渭　教授，博士生导师，主要研究方向为数据库与信息系统。

例1 设事务数据库 T 如表1所示，设最小支持度为2。项集 bcf 是频繁3项集，支持度为2，但不是闭合的，因为它有一个与其支持度相同的超集 $bcdf$。cdf 是闭合频繁项集。

表1 事务数据库 T

Transaction ID	Items in transaction
10	a,b,c,d,f
20	b,c,d,f,g,h
30	a,c,d,e,f
40	c,e,f,g

2.1.1 相关工作 目前已有很多闭合模式挖掘算法被提出，包括 A-close[7]，CLOSET[8]，MAFIA[9]，CHARM[10]和 CLOSET+[11]。A-close 使用宽度优先搜索挖掘频繁闭合模式。在密集数据库或有长模式的数据库中，因为大量的候选项集和多次的数据库扫描，宽度优先搜索会面临很大的困难。

CLOSET 算法是 FP-growth 算法的一种扩展形式，创建频繁模式树并自底向上地递归挖掘。虽然 CLOSET 使用了一些优化技术，但当数据库比较稀疏或支持度很低的时候仍然面临性能上的难题。

MAFIA 和 CHARM 都使用了数据库的垂直表示技术。MAFIA 主要用于挖掘最大模式，但也有可选项用于挖掘闭合模式。它的最主要的特点在于使用了压缩垂直位图结构(compressed vertical bitmap structure)。CHARM 采用了 dual itemset-tidset search tree 和 Diffset 技术。采用垂直结构算法中最耗时的操作是 tidset 的交集运算。

CLOSET+算法集成了之前提出的诸多算法的有效策略，同时提出了很多新的优化方法，比如混合树投影(hybrid tree projection)，项的跳过(item skipping)，子集检查(subset-checking)技术等。CLOSET+与 CLOSET 算法相比可以大幅度地提高性能，即使是在 CLOSET 和 CHARM 算法不能工作的时候也非常有效。

2.2 约束挖掘

一个约束是一个基于项集的幂集的谓词，$C: 2^I \to (r)\{\text{true,flase}\}$。一个模式 X 满足约束，如果 $C(X)$ 为 true。满足约束的模式的集合定义为 $SAT_C(I)=\{X \mid X\subseteq I \wedge C(X)=\text{true}\}$。如果一个模式属于 $SAT_C(I)$，则称之为有效的。

定义2(约束挖掘) 给定一个事务数据库 T，一个最小支持度 min_sup，和一个约束 C，基于约束的频繁模式挖掘问题定义为找出所有满足约束 C 的频繁模式 F_C，即 $F_C=\{X \mid X\in SAT_C(I) \wedge sup(x)\geqslant min\_sup\}$。

例2 仍以表1中事务数据库 T 为例，设最小支持度为2，约束 C 为模式中不含有项 f。则频繁模式 $bcdf$ 不满足约束，所以是无效的；而频繁模式 bcd 满足约束，是有效的。

基于约束的频繁模式挖掘已经研究了几年，有很多的研究成果，主要是基于约束的反单调性(anti-monotone)或者单调性(monotone)，或把约束加以转化，使其具有反单调性或单调性，以减少搜索空间，提高性能。

3 满足约束的闭合模式挖掘

要挖掘满足用户约束的闭合模式，由目前已有的工作，可以很直接地得出两种基本的方法。一种是首先对事务数据库运用闭合模式挖掘算法，挖掘产生所有的闭合模式，然后在结果集合的基础上扩展所有闭合模式得到全部频繁模式，再根据用户约束，去掉不满足用户约束的全部模式，得出满足用户约束的模式，然后去掉非闭合的模式。另一种方法是首先对事务数据库运用有约束的频繁模式挖掘算法，产生所有满足用户约束的频繁模式，然后在结果集合的基础上去掉不闭合的模式，只保留闭合的模式。用户可能已经有闭合模式的结果，但是想针对特定的约束查看模式，于是还要进一步得到满足约束的闭合模式；也可能已经有满足约束的频繁模式，但是数量巨大、不易理解和使用，于是希望得到闭合模式；也可能并没有中间结果，需要从头计算满足约束的闭合模式。甚至有的时候出于保密的原因不能得到原始的事务数据库，只能根据已有的闭合模式或者满足约束的模式进行加工，所以对以上几种情况分析如何高效地挖掘满足约束的闭合模式是值得进行研究的。

用户可以根据自己的目的灵活地提出约束，所以用户约束的形式是各式各样的。不同形式的约束对模式的挖掘算法的影响很大。下面首先把用户约束分为不同的几类，并主要对项约束讨论如何获得满足约束的闭合模式。本节仍然使用例1中的事务数据库 T，下面所有的讨论都是基于这个事务数据库的。

3.1 约束分类

用户感兴趣的约束主要有以下几类，包括项约束(item constraint)、长度约束(length constraint)、基于模型的约束(model-based constraint)、聚集约束(aggregate constraint)等[5]。虽然这些约束并不能概括所有的约束，但大多数约束都是在这几类中的。

项约束：项约束刻画了某项集是否包含或是否不包含于模式的性质。

长度约束：长度约束刻画了模式的长度性质。

基于模型的约束：基于模型的约束是寻找某特定模式(模型)的子集或者超集的模式。

聚集约束：聚集约束是基于模式中项的聚集函数的约束。

3.2 结合项约束的频繁闭合模式挖掘

下面把项约束分为几类，分别针对不同的类讨论如何进行满足项约束的闭合模式的挖掘。

3.2.1 约束为模式中不含某项或某几项 如果我们已经有闭合模式挖掘结果，则可以直接利用闭合模式挖掘结果得出满足约束的闭合模式。

例3 仍以表1中事务数据库 T 为例，设最小支持度 min_sup 为2，设 $C\equiv${模式中不含 d}。表2为事务数据库 T 中所有的频繁闭合模式，按照支持度分类。表3为 T 中所有的满足约束的闭合模式，按照支持度分类。

表2 事务数据库 T 中的频繁闭合模式，按照支持度分类

支持度	频繁闭合模式
2	cef,cfg,acdf,bcdf
3	cdf
4	cf

表3 事务数据库 T 中满足约束 C 的频繁闭合模式，按照支持度分类

支持度	满足约束的频繁闭合模式
2	cef,cfg,acf,bcf
3	
4	cf

算法如下：

```
按照支持度从大到小的顺序读取所有的闭合模式 p
{
    如果 p 满足约束 C
        把 p 加入内存中
        去掉所有被 p 吸收的模式
    如果不满足 C
        则改写 p，去掉不满足约束的项，标记为 p'
```

```
    将 p′依次与内存中已有的模式,标记为 p″,进行比较
    {
      如果 p′⊆p″,则跳出对 p′的检查(p′肯定非闭合)
      如果 p′⊃p″,则
        若 p′的支持度等于 p″的支持度,删除 p″
    }
    把 p′加入内存中
}
输出内存中的模式到输出文件中
```

算法正确性证明:

如果原有的闭合模式满足约束,则一定也是满足约束问题的闭合模式。

如果原有闭合模式不满足约束,去掉项之后是以前出现过的模式的子集,则可以直接忽略。因为闭合模式的读取是按照支持度从大到小的顺序,所以后读取的模式的支持度肯定小于等于前者。

如果不满足约束,去掉项之后是某个以前出现过的模式的超集,则支持度小于等于前者。如果等于,就删除前者。最后把这个模式添加到内存中。在算法结束的时候没有被删除的模式一定都是闭合的。

所以这个算法可以得出所有满足约束的闭合模式。

如果已经有满足约束的频繁模式挖掘结果,则可以直接从满足约束的频繁模式挖掘结果得到闭合模式。此时虽然用户已经得到所有的信息,但是对大量的模式找出闭合模式,可以缩小结果,对用户还是有用的。方法为从支持度高的模式开始,顺序读取每个满足约束的模式。检查以前的结果,是否有被吸收的情况。所有模式都读取之后把内存中的结果输出到输出文件中。在程序中可采用 CLOSET+中使用的 subset-checking 技术。

如果此前既没有闭合模式的结果,也没有满足约束的模式的结果,则直接对原数据库进行频繁闭合模式的挖掘,采用已有的闭合模式挖掘算法,如 CLOSET+,只需在扫描数据库时过滤掉那些不满足约束的项就可以了。

3.2.2 *约束为模式中必含某项或某几项* 比如约束可以定义为 $C\equiv${模式中必含 d}。

如果有闭合模式的结果:直接读取所有的闭合模式,检查是否满足约束,把满足约束的模式直接输出到输出文件中。

如果有满足约束模式的结果:采用与3.2.1节中相同的方法。

如果两者都没有:则对原数据库进行频繁闭合模式的挖掘,采用已有的闭合模式挖掘算法,只需在扫描数据库时过滤掉不满足约束的事务就可以了。

3.2.3 *约束为模式中不能同时出现某几项* 比如约束可以定义为 $C\equiv${模式中不能同时出现 c,d 和 f}。

如果我们已经有闭合模式挖掘结果,则可以直接利用闭合模式挖掘结果得出满足约束的闭合模式。

算法如下:

```
按照支持度从大到小的顺序读取所有的闭合模式 p
{
  如果 p 满足约束 C
    把 p 加入内存中
    去掉所有被 p 吸收的模式
  如果不满足 C
    则改写 p,去掉约束中 n 项中的1项,标记为 p′(共 n 个 p′)
    对每一个 p′将 p′依次与内存中已有的模式,标记为 p″,进行
    比较
    {
      如果 p′⊆p″,则跳出对该 p′的检查(p′肯定非闭合)
      如果 p′⊃p″,则
        若 p′的支持度等于 p″的支持度,删除 p″
    }
    把 p′加入内存中
}
输出内存中的模式到输出文件中
```

算法正确性证明与3.2.1节中证明类似。

如果有满足约束模式的结果:采用与3.2.1节中相同的方法。

如果两者都没有:则在建立频繁模式树时,对项所采用的排序中使 c、d 和 f 排在前三位。对频繁模式树进行挖掘中,当挖掘中若频繁模式基为 cd 时忽略结点 f,其他时候不需要做任何调整。

3.2.4 *约束为模式中至少有某项或者某几项* 比如约束可以定义为 $C\equiv${模式中至少有 d 和 f 中的一项}

如果有闭合模式的结果:直接读取所有的闭合模式,检查是否满足约束,把满足约束的模式直接输出到输出文件中。

如果有满足约束模式的结果:采用与3.2.1节中相同的方法。

如果两者都没有:忽略事务数据库中不含有约束中出现的项的事务,对每条事务进行排序,所采用的序使得约束中出现的项排在前面。只挖掘根结点的以 d 为根的子树中的频繁模式和根结点的以 f 为根的子树中的频繁模式。

4 性能分析

为了对前述算法进行性能评估,我们根据文[2]中所提供的事务数据库生成方法生成了几个事务数据库进行实验和性能比较。事务数据库用 Tx.Iy.Dz.Nw 表示,其中 x 表示平均事务长度,y 表示平均最大模式长度,z 表示事务数据库中事务的条数,w 表示事务数据库中项的个数。图1给出了用前面3.2.1节中所讨论的算法对 T25.I20.D100k.N1k 和 T25.I10.D10k.N1k 两个事务数据库的闭合模式结果进行第一类约束挖掘处理的实验结果。约束中的项为一项,约束中的项多于一项的情况和3.2.3节中所讨论的算法和该算法性能类似。对于不是非常密集的事务数据库,由实验性能曲线可以看出,我们的算法是有效的。

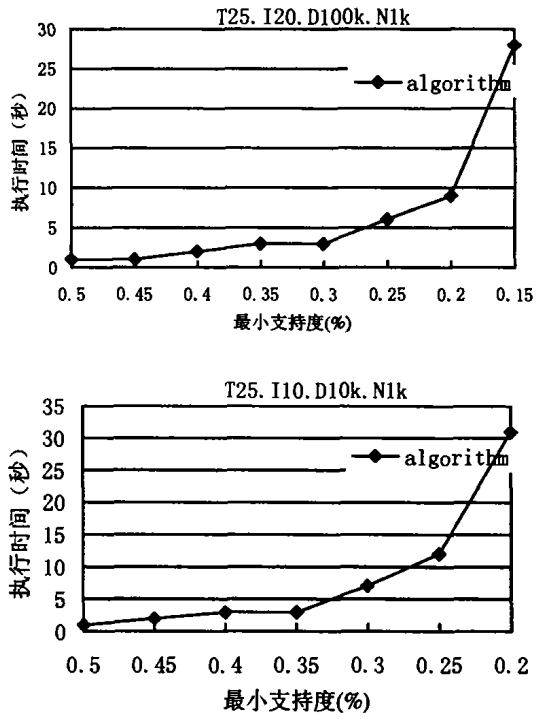

图1 部分实验结果

结束语 本文在频繁闭合模式的挖掘和满足约束的挖掘研究的基础上,主要讨论了项约束的分类,并根据不同的项约束研究了频繁闭合项集挖掘算法,实现了算法,给出了性能分析。实验表明我们的算法是有效的。如何结合长度约束、基于模型的约束、聚集约束等其他形式的约束,以及如何使闭合模

式的挖掘和满足约束的挖掘结合得更为紧密是我们今后所要研究的问题。

参 考 文 献

1 Agrawal R, Imielinski T, Swami A. Mining Association Rules between Sets of Items in Large Databases. In: Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93), Washington, DC, MAY 1993. 207～216

2 Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules, VLDB'94, 487～499

3 Park J S, Chen M S, Yu P S. An Effective Hash-based Algorithm for Mining Association Rules. In SIGMOD'95, 175～186

4 Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation. In: Proc. ACM SIGMOD, 2000. 1～12

5 Pei J, Han J. Can We Push More Constraints into Frequent Pattern Mining? In: Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), Boston, MA, AUG. 2000. 350～354

6 Pei J, Han J, Lakshmanan L V S. Mining Frequent Itemsets with Convertible Constraints. In: Proc. 2001 Int. Conf. on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001

7 Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering Frequent Closed Itemsets for Association Rules. In ICDT'99, Jan. 1999

8 Pei J, Han J, Mao R. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In: Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX, May 2000

9 Burdick D, Calimlim M, Gehrke J. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In ICDE'01, April 2001

10 Zaki M, Hsiao C. CHARM: An Efficient Algorithm for Closed Itemset Mining. In SDM'02, April 2002

11 Wang J, Han J, Pei J. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In: Proc. 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03), Washington, D. C., Aug. 2003

(上接第113页)

时发布[18]，事实上两款工具的开发者经常探讨交流他们的技术，期望最后两个工具能融合一体。

RATS 比 Flawfinder 支持更多的语言，提供了对 C、C++、Perl、PHP 以及 Python 语言的漏洞扫描。RATS 也是遵循 GPL 协议开发的开放源码的工具。

4.2.4 BOON BOON(Buffer Overrun detectiON)是加州伯克利 David Wagner 博士论文的实现原型，是2002年发布一款专门针对 C 程序的缓冲区溢出漏洞检测工具。

BOON 在实现内部借鉴面向对象思想，对每个字符串变量设置了两个属性，一个描述该字符串被分配的大小，另一个描述字符串实际被使用的大小。所有的字符串处理函数都将参考这两个属性对字符串进行操作。BOON 通过扫描计算后得出源程序中的安全漏洞级别，程序员根据 BOON 的报告然后手动改正源程序的缺陷。

BOON 不检查有关格式化串(Format String)漏洞的扫描，而且现在作者也明确表示不再有更新版本出现。

结论 尽管缓冲区溢出存在了多年，并且业界也对缓冲区溢出展开了不少研究，但问题还是没有得到根本的解决，缓冲区溢出仍然是计算机安全所面临的重大课题。

在我们的总结中发现，无论是动态检测技术，还是静态发现技术，都或多或少地减少了缓冲区溢出攻击的可能。我们尚未发现哪一种技术是完美的，当然我们分析的也是其中的一部分比较有代表意义的解决方案。在动态检测技术中，还有其他的如针对格式化串的攻击检测 FormatGuard、硬件技术和编译技术结合的 StackGhost、增强 C 的类型和边界检查的 Ccured 和 CyClone 等，以及在静态发现技术中的，对程序安全属性进行建模分析的 MOPS、对 C 类型增强的 CQUAL、对 Unix 平台 Lint 进行安全扫描改进的 Splint 等，限于篇幅我们没有分析。不过解决问题的角度基本都从编译器或者操作系统方面进行探索。

假设每一个对缓冲区的操作都得到严格的检查，假设根本就没有溢出发生，那就不会有缓冲区溢出的攻击。如果我们的高级语言能保证不会发生越界情况，那这个问题也就能得到解决。但计算机已经有长达半个多世纪的历史，现在遗留下来的软件是宝贵的财富，我们不可能做到所有的程序都重新编译，所以解决的办法都尽量地从兼容性、从最小化改动原有程序考虑。同时，各种各样针对缓冲区溢出的新的攻击手段可能也会出现，这些都是缓冲区研究的难点。

今后研究的重点可能会集中在如入侵检测技术、程序自保护技术、系统体系结构研究等方面，我们也正是从这些方面着手的。完全地走出缓冲区溢出阴影，可能还需要研究人员作出更多努力。

参 考 文 献

1 Cowan C. Buffer Overflow and OS/390. http://cert.uni-stuttgart.de/archive/bugtraq/1999/02/msg00081.html

2 CERT, the Computer Emergency Response Team Coordination Center. http://www.cert.org/advisories/

3 Executable and Linkable Format (ELF). Portable Formats Specification, Version 1.1

4 Fayolle P A, Glaume V. A Buffer Overflow Study Attacks & Defenses. ENSEIRB, Networks and Distributed Systems, 2002

5 Wilander J, Kamkar M. Dept. of Computer and Information Science, Link¨opings universitet. A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention. 2002

6 Chiueh T, Hsu F H. Computer Science Department, State University of New York at Stony Brook. RAD: A Compile-Time Solution to Buffer Overflow Attacks. In: Proc. of The 21st IEEE Intl. Conf. on DISTRIBUTED COMPUTING SYSTEM, 2001. 409

7 Conover M. w00w00 Security Team. w00w00 on heap overflows. http://www.w00w00.org/files/articles/heaptut.txt, Jan. 1999

8 Cowan C, et al. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In: 7th USENIX Security Conf. San Antonio, Texas, USA, 1998

9 GrSecurity. http://www.grsecurity.net/

10 Cowan C, et al. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In: Proc. of the DARPA Information Survivability Conf. and Expo (DISCEX), Hilton Head, South Carolina, Jan. 2000. 119～129

11 Jones R, Kelly P. Backwards-compatible bounds checking for arrays and pointers in C programs. In: Proc. of the Third Intl. Workshop on Automatic Debugging AADEBUG'97, Linkoping, Sweden, May 1997

12 Bulba and Kil3r. Bypassig StackGuard and Stack- Shield. Phrack Magazine 56 http://www.phrack.org/phrack/56/p56-0x05, May 2000

13 Cowan C, et al. PointGuard: Protecting Pointers From Buffer Overflow Vulnerabilities. In: 12th USENIX Security Symposium, Washington DC, 2003

14 Vendicator. Stack Shield technical info file v0.7. http://www.angelfire.com/sk/stackshield/, Jan. 2001

15 Etoh H. GCC extension for protecting applications from stack-smashing attacks. http://www.trl.ibm.com/projects/security/ssp/, June 2000

16 PaX http://pageexec.virtualave.net

17 Viega J, et al. ITS4: A static vulnerability scanner for C and C++ code. In: Proc. of the 16th Annual, Computer Security Applications Conf. Dec. 2000

18 Wheeler D A. Flawfinder, Web page http://www.dwheeler.com/flawfinder/, May 2001