基于语法解析树的函数漏洞发现方法

陈永艳[1] 束洪春[2] 戴 伟[3]
(昆明理工大学计算中心 昆明 650500)[1] (昆明理工大学研究生院 昆明 650500)[2]
(昆明理工大学省计算机技术应用重点实验室 昆明 650500)[3]

摘 要 大多数行业定制软件的漏洞检测较困难,而传统的静态漏洞检测方法会产生很多错误的和虚假的信息。针对函数调用前后存在的漏洞问题,提出了基于上下文无关的自顶向下与自底向上相结合的语法解析树的方法,它能够在对函数内部定义不了解或者部分了解的情况下,解析函数调用前后安全契约规则:前置规则和后置规则。同时通过扩展规则表示的XML文法来表示面向对象下,规则中的属性存在继承关系下的契约规则。实验表明,与同类型安全分析工具比较,该方法具有避免函数重复分析、规则可扩展性良好、尤其在自定义对象类和特定环境下自定义参数准确率高等优点。

关键词 函数弱点,继承关系,契约规则,语法解析树
中图法分类号 TP311 **文献标识码** A

Function Vulnerability Detection Method Based on Parse Tree

CHEN Yong-yan[1] SHU Hong-chun[2] DAI Wei[3]
(Computer Center, Kunming University of Science and Technology, Kunming 650500, China)[1]
(Department of Graduate, Kunming University of Science and Technology, Kunming 650500, China)[2]
(Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China)[3]

Abstract Custom software vulnerability detection is difficult. Most of static vulnerability detection approach usually produces large amount of false information and positives results. A new method is able to understand the analyzed source code when a function is called. This paper proposed a method of combination top-down and bottom-up parsing tree which is based on CFL(context-free language). In a case of not understanding or partially understanding inside code of a function definition, it can analyze function contract before or after function called, named pre-condition and post-condition. Extending the rules of XML grammar on object-oriented, pre-condition and post-condition can deal with objects belonging to inheritance relationship's class. The experiments show that, compared with the same type of security analysis tools, it can avoid repeat function analysis, has good rules scalability and high accuracy for custom defined object classes and parameters in custom environmental especially.

Keywords Function vulnerability, Inheritance relationship, Contract rules in inherent, Parse tree

1 引言

和通用软件相比,排查由软件设计产生的漏洞更为困难。通常,软件脆弱点可被定义为软件系统中一个不好的特性或部分,该特性可能导致潜在威胁,这种威胁可能是恶意的或是能对计算机系统相关资源产生不期望的影响的。软件代码内部自身的缺陷可被一些恶意的用户用来引发各种安全问题,如获取非授权权限,干扰关键进程的执行等。这一类问题无论是在通用软件还是专用软件上都不可避免,若连接在应用程序上,会更加突出。目前已发现的代码内部存在的脆弱性类型繁多,如缓冲区泄露、格式字符串、条件竞争、SQL注入等。针对该类脆弱点,国内外许多学者也提出了很多定位和识别方法,其中包括侵入分析、静态分析、模型校验等,同时对脆弱点分类也作了大量工作[4,5,11]。

随着软件系统的规模和复杂性逐渐增长,威胁软件运行的漏洞和设计存在的缺陷也随之增长。在电网信息网上运行的众多专用软件,并非像大多数Windows系统、Java、Linux或者Oracle众多软件一样会定期地公布漏洞并即时提供漏洞补丁修复。针对专用软件弱点进行识别和修复,需要软件开发人员和使用人员经过一定的测试和使用期,进行大量检测和排查来确定并修复漏洞。为了防止这些问题的扩展,目前漏洞检测技术已经被广泛地集成到软件开发过程。通过对大量的已公布的CVE安全漏洞进行分析,可以看出,大多数安全攻击均是利用的源代码的弱点。

在C语言中最常见的缓冲区就是字符数组,而操纵字符数组的函数有gets、strcpy、sprintf等都存在一定的漏洞。很

到稿日期:2012-10-16 返修日期:2013-02-04 本文受国家自然科学基金项目(11103005)资助。
陈永艳(1975-),女,硕士,讲师,主要研究方向为电网信息系统安全评估,E-mail:dawanmitang@163.com;束洪春(1961-),男,博士,教授,主要研究方向为电力系统继电保护和电力系统稳定控制,E-mail:shuhongchun@kmust.edu.cn(通信作者)。

多这类函数在执行字符串拷贝的过程中没有对字符串进行长度检查，这样就很容易发生超长的字符串溢出缓冲区的情况。当初这样设计是出于效率的考虑，但现在看来，这些函数的使用已成为C语言软件脆弱的一个重要因素。如果程序员没有良好的编程习惯，没有时刻注意函数调用过程中是否拷贝了超过缓冲区长度的字符串，那么缓冲区溢出就不可避免。

例如对于下面一小段代码：

```
#define BUFFSIZE 8
int main(int argc,char * argv[])
{
char buff[BUFFSIZE];
strcpy(buff,argv[1]);
}
```

如果用户输入了超长的字符串，除了填满了缓冲区，还覆盖了其他一些程序正常退出所需要的数据。检查strcpy的算法(部分)定义如下：

```
char * strcpy(char * dst,char * src)
{
  char * cp=dst;
  while( * cp++= * src++);
  return(dst);
}
```

很明显这段定义是有漏洞的：

(1)没有检查输入的两个指针是否有效。

(2)没有检查两个字符串是否以NULL结尾。

(3)没有检查目标指针的空间是否大于等于原字符串的空间。

同样，这样使用strcpy(buf,"test. data. txt");}也是有问题的，如果输入来自一个无效的源，它同样会导致程序崩溃。

以上是一个典型的缓冲区溢出的安全漏洞。早在20世纪80年代，缓冲区溢出就已经为人所知，但时至今日，大量的缓冲区溢出漏洞仍层出不穷地被发现。最著名的Morris蠕虫就是利用Unix系统上fingerd程序的缓冲区溢出漏洞。在Oracle 9i发布之初，Oarcle公司曾宣称他的数据库是"unbreakable"的，但不到几个月的时间，就暴出Oracle 9i中oracle. exe、XDB等程序存在多个缓冲区溢出漏洞。

对于这类的弱点问题，可以采用渗透性测试来完成。渗透性测试的基本方法包括：黑盒、白盒、灰盒。常用的渗透性测试工具包括Metasploit、nessus、Nmap、webinspect、Immunity CANVAS、Core Impact等等。但基本上，渗透性测试工具只针对CEV等公布的已知漏洞采用端口扫描方检测进行排查，其检测也多属于黑盒测试，尤其针对Web漏洞。例如webinspect通过截获浏览器端的http request和http response，然后通过修改内容参数和编码来伪造请求，按照一定规则重放请求，借此发现SQL注入漏洞。这些高层次的分析工具很少有针对行业定制的软件进行排查，而且排查结果往往由于其模式不具备一定的语法结构，很难分析和定位具体语法下软件编程造成弱点的关键部分。当然有些静态工具的功能，例如：Code-Check[2]、FlawFinder[3]等程序员以整个项目为单位提交待检查代码，以整个项目代码为单位执行安全检查，并根据语法分析并报告大量的错误，但其实这些错误很多都是误报，并不是真正的错误，浪费了很多程序员的时间和精力。Coverity[1]解决了影响源代码分析有效性的4个关键问题：构建集成、编译兼容性、高误报率、有效的错误根源分析。Prevent分析引擎中有两个针对跨过程分析的功能：一个是过程间调用总结引擎，另一个是类型流程引擎。前者使得Prevent可以执行整个程序的分析，分析文件间和模块间的复杂的调用链，后者用于分析依赖于类层次关系的报告。在实际的行业专业软件当中，对于可能产生漏洞的代码部分的危害安全级别要求有所不同。例如在电力信息网文件的传输中，为了减少内部攻击窃取数据，根据容入侵的级别要求，可以采用FRS技术将知识信息(数据、程序)分片并分散存储到不同的节点，这就需要根据安全等级情况对代码进行分析。很明显调度网和营销网的安全级别设定和隔离要求不同，对容入侵的级别要求也会不同，从而对于这两部分代码的分析可能存在差异。所以在本文的方法中，将这两部分分析功能整合，更增加了用户自行设定契约规则的方式，以针对软件在特定环境下的需求进行调整。

在很多行业专用软件投入使用之前的试用期期间，针对代码本身造成的漏洞的排查大多属于综合白盒和黑盒二者兼顾的测试方式，测试的工作量相当繁重耗时。在此种情况下，希望能有一个工具，可根据安全规则主动深入了解源代码，并在编程过程中，寻找那些不属于语法错误但会带来威胁的弱点部分，并定位这些弱点所在的代码。为了达到这个目的，本文目前的研究，提供了新的静态函数调用安全漏洞探查解决方案。此工具属于一种针对函数调用灰盒测试方式的安全漏洞测试工具，其能够在对函数内部定义不了解或者部分了解的情况下调用函数，给出函数调用前后安全使用规则。此工具属于提取外层次的安全规则的编程概念，考虑在类继承条件下，利用语法解析树的基于上下文无关的自顶向下与自底向上相结合的方式解析XML规则文法，主要用于检测软件代码中的函数调用前后代码部分是否存在漏洞，尤其是缓冲区溢出漏洞。

2 函数被调用前后的契约规则

本文中全局项目代码分析检查属于跨过程分析。传统的跨过程分析必须首先判断函数之间的调用和被调用关系。可以通过构造控制流图、数据流图得到程序的控制依赖和数据依赖图，来构造出系统依赖图(system dependence graph，SDG)。系统依赖图中记录了所有的函数调用层次和数据依赖关系。当系统复杂度和规模增大时，系统依赖图的复杂度和规模会相应增大，对函数漏洞的考查也变得十分困难。传统的是按照程序执行顺序来分析，这明显会对多次调用的同一函数重复分析。为了避免这一问题，须了解函数之间的调用层次关系，并采用自内而外的解析函数体的方式；必须对每一个函数采取单独分析的方式，考核函数调用前满足的安全规则和调用后对环境的影响规则。

引入契约机制[7]表示函数调用前后的安全规则，避免重复分析函数，在对函数内部代码未知或者半未知情况下实现跨过程分析。假设本文中的契约机制分析函数之间的调用不存在死锁和递归调用。

关于契约机制的使用，Hoare逻辑是个常用的方法。文献[16]给出实现安全调用的Hoare逻辑下的逻辑指针，使用其Hoare逻辑来描述过程调用，此方法将函数当作一个抽象的单元从源代码中分离，根据源代码中的调用关系推导出前

置条件和后置条件，着重解决全局变量作为地址调用参数时的条件推导。但此方法有一个问题未解决，就是在存在一定继承关系的参数，即祖先类和子孙类之间前置或者后置条件相互影响。同时文中并未给出后置条件中返回值影响的过程。

函数之间的调用依赖关系表示如下：按照由内而外的解析过程解析函数调用关系时，B必定先于A分析，即调用B之前为保证B安全执行所需的前后环境检查和调用B之后B对A的前后影响记录两部分。将安全检查所需的函数上下文信息定义为函数契约规则。对所有调用函数B，为保证B安全执行，必须满足一定的环境条件。前后环境包括两部分：第一部分是执行函数B的全局变量与参数必须满足的全部状态集合，称为函数B的前置规则PreCondition(B)；当函数B被调用完成之后对主调函数产生的影响，包括全局变量、参数和返回值的全部状态集合，称为函数B的后置规则PostCondition(B)。函数B的前置规则和后置规则合并称为函数B的契约规则。其中函数B的后置规则PostCondition(B)的安全规则中还需要增加返回值对主调函数A的影响。

函数调用关系漏洞分析的方法自内而外展开，假设函数A调用函数B，分析安全执行B所需的安全调用规则的前置规则和后置规则，再返回A函数检查，主调函数A的前后文环境是否满足执行B的前置条件，并计算B的后置条件对后文所带来的影响。契约规则执行流程如图1所示。

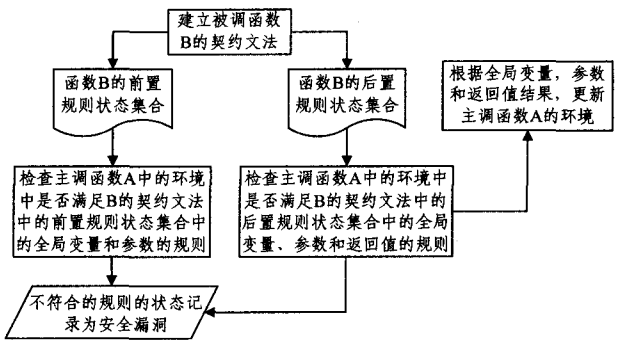

图1　契约规则执行流程图

考核函数调用契约规则分两类：系统函数和自定义函数。首先来分析系统函数部分。

这种未截断的字符串会泄漏内存信息。还有一些库函数也不会自动截断字符串，如fread()、read()、ready()、pread()、memcpy()、memmove()、bcopy()等，表1给出了常见的几个字符处理函数的契约规则。

表1　常见函数的契约规则

弱点模式	前置规则	后置规则
strcpy(dst,src)	src. Size<=dst. MaxSize	dst[MaxSize]='0'
strncpy(dst,src,n)	Min(src. Size, n)<=dst. MaxSize	dst[MaxSize]='\0'
strncat(dst,src,n)	dst. Size+Min(src. Size, n)<=dst. MaxSize−1	dst[MaxSize]='\0'
strcat(dst,src)	dst. Size+src. Size<=dst. MaxSize)	dst[MaxSize]='\0'
gets(s)	∞<=s. MaxSize−1	dst[MaxSize]='\0'
memcpy(dst,src,n)	Min(src. Size, n)<=dst. MaxSize	dst[MaxSize]='\0'

3　函语法解析树

每一个函数的弱点需要定义其前置规则和后置规则，完整的规则将以XML文法形式保存，调用函数代码部分需要对被函数调用的调用安全规则进行检测分析和匹配，最终给出结论是否属于安全调用。本文将引入语法解析树来完成匹配分析的过程。通过对每个函数的多个脆弱性设置使用规则统一的模式描述来约束，即使用简单的抽象概念描述，就可以有效减小匹配分析过程的盲目试探性，从而提高检测分析的效率。本文提出了利用语法解析树方法来解析XML文法的契约规则。

形成契约规则文法库，先将同一类弱点规则和约束用指定的XML文法模式描述，所有弱点类型的文法(即模式)组成弱点文法库。在此基础上，利用语法解析树解析XML文法中的规则库，来判断是否违法规则。由于契约规则的文法为由内而外的解析，对应于解析树则可采用与上下文无关的自底向上的解析树方式。但自底向上的解析策略中存在一个致命的缺陷，即解析的过程会执着于那些没有希望的相邻的结点，会毫无控制地滋生出无关的解析结点。而自顶向下的解析不会浪费时间去搜索不可能的树枝，但也会花费大量时间努力去查询输入不一致的分支。为了很好地解决这个问题，把二者结合起来使用。解析树的树根只生成两个分支：左分支记录漏洞属性，右分支记录漏洞契约规则。其解析的方法是：先对左分支自顶向下解析来得到漏洞属性建立属性栈，再采用自底向上的方法对应解析契约规则并形成契约规则栈。

将两种解析方法结合，需要对原有的两种解析树的解析算法做些调整和修改。采用深度优先策略方式，首先对根的左结点子树递增地展开空间搜索过程，搜索算法首先完成搜索漏洞属性状态。根据搜索出的漏洞属性状态结点，从树根的右半边结点采用右递归方式选择展开的状态解析，如果深度优先方法达到的树与指定状态结点符号串不一致，返回到新生成的但未被搜索过的树继续进行，直到找到为止，执行入栈操作。

Attribute Stack表中表示漏洞的属性原型，每个属性原型表示一个搜索状态。每个搜索状态由局部树和指向句子中的下一个叶子指针组成。搜索的过程是一个回路循环过程loop，循环会从Attribute Stack中找一个未被搜索的状态集，并应用相应可用的全部语法规则，把这个新状态集链接到Attribute Stack的最前面。反复重复以上过程，直到发现成功的解析树或者无法完成解析树，即搜索状态表的结果为空。解析树算法如图2所示。strcat()对应契约规则文法完成的语法解析树，如图3所示。语法解析树的第一部分为漏洞属性原型部分，包括函数所用的全部全局变量、参数和返回值分别解析；第二部分为契约规则解析。根据全局变量、参数和返回值满足的函数依赖与契约关系分别定义，同时根据解析算法形成满足的属性规则库的前置契约和后置契约。如果解析完成之后，仍然存在无法完成的解析树，例如规则里用到的属相原型并未定义，则认为契约规则定义有误，需重新定义契约规则，以保证契约规则所用到的属性原型全部在语法解析树的函数属性栈中全部定义过。

Input: condition node

Output: complete parse tree or not commplete parse tree, and build attribute stack and contract rules stack

Initialization: Attribute Stack and S tree

```
POP(Attribute Stack)－> currrent_search_state
loop
  if parse_tree is successful(currrent_search_state) then
      return parse_tree(currrent_search_state)
  else
      if CAT(Node_to_Expand(currrent_search_state)) is a POS
then
      if CAT(Node_to_Expand) belong to POS(Current_input(cur-
rrent_search_state))
then
      PUSH(Contract Rules Stack (currrent_search_state,S)
        else return reject
        else PUSH(Contract Rules Stack (currrent_search_state,S)
        if (Attribute Stack is empty) then
          if (S tree is not empty) then
            return reject
        else
            NEXT(s)－>currrent_search_state
end
```

图 2　解析树算法

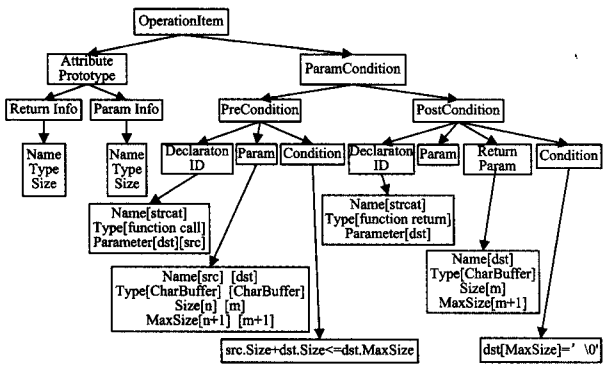

图 3　strcat()函数的语法解析树

最后根据前置契约和后置契约便可开始漏洞检查。根据漏洞检查结果判断该节点为弱点节点，存在漏洞，生成弱点分析报告。

4　契约规则匹配

建立好契约规则之后，开始检查的函数漏洞的问题。针对语法解析树的每一个结点设置的函数调用前后的每一条契约规则完成匹配，以判断其规则是否满足。匹配的过程与其他例如入侵检测有所不同，函数调用的契约规则匹配存在以下特点：

(1)多条契约规则匹配存在多路分支。因为函数漏洞查询的是符合语法定义但逻辑有错误或者对属性原型不安全的使用，针对参数、全局变量和返回值生成的规则可能存在多条，并且规则之间存在一定的逻辑关系，可能存在根据查询条件结果来匹配。

(2)每条规则必须解析。在漏洞判断的过程中，必须解析每条规则，经过解析后给出结果是符合或者不符合规则，从而给出该函数存在的全部漏洞问题。

(3)契约规则中涉及到的属性原型，包括参数、全局变量或者返回值，可能存在一定的继承关系，即为对象实例，对象实例的属性和方法都存在一定的继承关系，必须考虑其与祖先类或者子孙类的相互影响。

多规则和条件规则的情况下，XML 数据查询多属于多分支路径查询。文献[10]提出了 pathjoins 算法来解决多分支查询，其基本思想是：首先对查询树进行编码，将其拆分成由根结点到叶子结点的单路径，并用 pathstack 算法得到单路径查询结果，然后将单路径查询结果利用结构连接算法进行匹配得到多分支查询结果。本文中借鉴此方法，但对方法做了些的变动，扩展 XML 文法以使其支持表示祖先类和子孙类属性和方法的继承关系，以及可以表示出由继承关系产生的影响。

为了加快的匹配过程，必须对文档进行编码并创建索引，然后对查询树进行查询匹配。采用栈存储多分支路径中的单路径，对多分支结点采用索引信息判定其子结点是否具有共同的祖先结点或父结点。如果匹配规则较多，在内嵌条件存在嵌套条件的情况下，查询就变得十分复杂。因为 XML 文法形成的 DOM(Document Object Model)查询树如果想以更快的方式找到其父节点，则在文献[10]查询匹配算法中进行高效的路径连接，树各个结点采用类似于有类 IP 地址的编号方式。设查询树根结点的编号为"0"，其孩子结点的编号由其父结点的编号和其属于其父结点的第几个孩子结点组成。

但此种查询存在一个问题，如果多规则条件中的参数之间相互存在继承关系，而且参数之间的继承关系还有多重继承，针对父类的和子类的属性的查询条件的关联就不是一对多的关系。如果表达这类关系，这种方法就不能反映出继承过程中属性原型的关联结果。

将 XML 扩展至表达对象的单继承和多重继承关系，则需要使用 XML-RL 使用语言[6]来定义属性原型。XML-RL 专门的 XML 文法来表示存在继承关系的类，尤其解决了多重继承下继承关系的表示。但是 XML-RL 有两个问题没有解决：一个问题是并未对类的属性和方法的使用给出详细说明，属性和方法是在祖孙类中定义，还是在本类中定义；第二是重写 Overriding 和重载 Overloading 的函数的确定没有说明。Overriding 属于父类和子类同名函数的情况需要遍历祖先类来确定；而 Overloading 因为属于一个函数的多态表现，只要通过参数就可以判断。下面给出一个典型的自定义函数契约规则来说明这个判断过程。

电网的呼叫中心用来响应客户的包括公共信息访问、网上缴费、来电信息采集充放电储能管理等众多业务。假设用户从终端呼叫，坐席代表将给予响应。所以用到的类为：AnswerSheet 类继承 Sheet 类，这是一个单继承，一个是用户提交的填表，一个是坐席代表回复填表内容；Guest 类和 Receptionist 类继承 User 类，这个也是单继承。Guest 类是远程访问用户，Receptionist 类表示坐席代表。Tuser 类用来表示特殊身份的管理员，从远处登录进来，但需要对 Tuser 远程访问登录是控制和限制其操作权限。所以 Tuser 类同时继承了 Guest 类和 Receptionist 类，这是一个多重继承。Guest 类、Receptionist 类和 Tuser 类都有 takes 方法，但 Guest 和 Receptionist 都重写了 takes 方法。pid 和 name 为 Tuser 定义，sid 为 Sheet 定义。

首先通过 DOM 解析器将 XML 文档解析生成 DOM 树，并由 DOM 树生成树形结构的数据树。对数据树进行先序遍历，给每个结点依次编号，合并相同对象结点，相同结点的路径表示为一条标记路径，生成查询树。

接下来需要用类之间的继承关系来修改查询树的概要路径表示契约规则中属性原型的继承关系。修改路径概要中的

每个组的相当结点的值，合并继承关系下继承而来的属性，但保留本类中单独定义的属性，生成面向对象 XML 数据树，如图 4 所示。

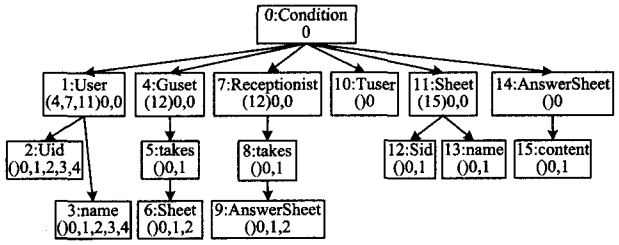

图 4　strcat()函数的语法解析树

调用 TagTable 映射表匹配查询树的结点，返回相应的结点值。用 *size* 表示 Uid 的后代组的数量，g_c 表示子孙结点组，g_f 为祖先结点集合。由祖先类和后代类的继承关系，执行连接操作。$g_f.Uid < g_c.Uid \leqslant (g_f.Uid + g_f.size)$ 和 $g_c.level \geqslant g_f.level + 1$。根据查询结果继续判断如果查询为叶子结点，也就是所需的查询结果，则根据头信息表给出其所在组的范围。

如果不满足条件，则需要重新判断二者之间的关系。必须从基类查起，以判断其所属的祖先的继承关系。现需建立两个信息表，一个是基类信息表 BaseClassTable{Uid, baseClassUid}用来表示某类 Uid 的父亲 baseClassUid；另一个是子类信息表 ChildClassTable{Uid, childClassUid, childIndex}，childClassUid 用来表示子类，childIndex 表示子类所在组中索引值或者索引范围值。根据这两个表搜索的结果，可以判断出所要查找的两个对象是否存在继承关系。

接下来开始完成匹配过程。查询语句部分将包含两部分：结构约束和谓词约束。结构约束根据以上所述的继承关系查询方法来完成，谓词查询从叶子结点开始，确定叶子所处组(类)中的索引值，并将查询结果向上投影求出交集，最终完成全部查询。

例如将判断上例完善其前置规则。本例中的规则条件是所有 User 的全部子类，包括：Guest、Receptionist 和 Tuser 中的子类，其中排除 $p 中所指类别，查找满足条件的 User 其 name 属于 $a 的对象 answerSheet 的名称。

假设前置规则条件如下：

```
〈PreSubCondition〉
    /precondition/user→[@Uid→$u,name→$n]
〈/PreSubCondition〉
〈PreSubCondition〉
    /precondition/user↑→[@Uid→$p,name→$n]
〈/PreSubCondition〉
〈PreSubCondition〉
    Tuser→@ta
    for_each $b//answerSheet→$b exit $c//(answerSheet)con-
    tent→c$a/name→@queryName such that ($b/@Sid=$c)
〈PreSubCondition〉
```

第一组前置条件中 User 的子类 Guest、Receptionist 和 Tuser 的 Uid 和 name 必须全部包括进去。第二组前置条件中 User 查询子类过程中属于独占元素，必须排除 $p 所指类的实例。第三组前置条件中为多态引用。Tuser 继承 Guest、Receptionist 而来，answerSheet 属于 sheet 子元素，answerSheet 为 Tuser 继承 Receptionist 属性而来。查询过程为遍历每一个 Tuser 中的满足条件的 $Sid = \$c$，得到 Tuser 的属性 answerSheet. content 的结果。

后置规则条件如下：后置条件为返回值结果，结果将返回在调取 answerSheet 的操作中，属于修改 answerSheet. content 的用户和被修改 answerSheet 变动的 Sid。

```
〈PostCondition〉
/results/Return→[@Uid=@p,name→$n, $b/@Sid=$c)
〈/PostCondition〉
```

使用以下两个指标来评判弱点检测工具检测结果。

false positive 分别表示平均失败比率：

$$false\_positives = 1 - TP/NUM$$

其中，*TP* 代表真正的弱点检测工具的项目的数量，*NUM* 表示检测工具的输出结果条目的数量。

$efficiency = TP/NUM$ 用来表示检测效率。

测试用例中的安全漏洞根据 NASA-GB-8719.13 中的部分漏洞要求编写契约规则而成。测试时读入包含安全漏洞函数的用例代码的单个源文件，进行安全分析。经测试，安全检查工具能够实现所有测试用例中包含安全漏洞的正确检查。针对用户自定义的函数的集成测试，依次读入源程序分析每个函数构造，依据调用关系构造函数依赖图，确定函数分析排序顺序，根据函数的调用和返回规则编写生产前置规则和后置规则，对每个函数完成安全分析，并报告分析结果。

本次测试选取了电网信息网的成熟的级配的旧版本软件产品。该软件系统(A,B 和 C)是基于 C++的，并作为服务器。该项目有不同的发展进程。源代码还包含了一些已知的漏洞，这些漏洞已出现在前期的开发和测试团队报告中。A 代码包为营销系统中有序用电执行情况统计，大约 1 万行代码量。代码库包含了一些第三方的控件代码。B 代码包大约有 6 千行，主要功能是信息渡船中完成文件和数据的上传。为了避免数据泄漏，对部分文件发送的格式采用动态分片的 FRS 技术。C 代码包产品有 5 千行代码，针对有序用电数据的部分功能，主要任务是提供数据给最终用户。

表 2　静态代码测试结果比较

实际漏洞数量		Coverity Prevent	Flawfinder	语法解析树的漏洞发现方法
Input validation and representation	A 5	5	5	2
	B 3	5	3	2
	C 3	3	3	3
API abuse	A 0	1	1	0
	B 2	4	2	2
	C 1	1	1	0
Security features	A 1	1	1	1
	B 0	0	0	0
	C 0	0	0	0
Time and state	A 0	0	0	
	B 0	0	0	0
	C 0	0	0	0
Errors	A 4	0	0	4
	B 5	0	0	5
	C 4	0	0	4
Code quality	A 7	23	15	4
	B 5	9	5	4
	C 5	6	5	1
Encapsulation	A 4	1	0	4
	B 5	1	0	5
	C 5	0	0	5
Environment	A 0	0	0	0
	B 0	0	0	0
	C 0	0	0	0

(下转第 135 页)

略，为服务的响应时间提供了有力的保障。

云服务通过动态协作的网络资源以一种柔性可演化、连续反应式、多目标适应的组合新形态实现了服务资源的有效动态配置和共享使用，为跨企业、跨组织的分布式应用系统的构建提供了支持。云服务网络环境的开放性、平台的异构性以及通信的异步性和服务构件的自治性等不确定因素使得云服务组合的可靠性受到了巨大影响，这使得云服务的可靠性优化必须考虑每个时刻组成云应用服务的实时可靠性状况。下一步的工作重点是基于现有的云计算服务应用平台，从不同角度考虑影响云服务可靠性的因素，并进行应用实例分析，进一步优化可靠性设计方法，真正达到为云服务用户提供持续可靠的运行服务的目的。

参 考 文 献

[1] Jayasinghe D. FAWS-a client transparent fault tolerance system for SOAP-based Web services[EB/OL]. http://www-128.ibm.com/developerworks/webservices/library/ws-faws/,2005

[2] Liang D, Fang C L, Chen C, et al. Fault-tolerant Web service [A]// Proceedings of the 10th Asia2 Pacific Software Engineering Conference(APSEC'03)[C]. ChiangMai, Thailand, 2003:310-319

[3] Santos G T, Lung L C, Montez C. FTWeb: a fault tolerant infrastructure for Web services[A]// The 2005 Ninth IEEE International EDOC Enterprise Computing Conference(EDOC'05)[C]. Enscheda, Netherlands, 2005:95-105

[4] Lin M, Chang M, Chen D. Distributed-program reliability analysis: complexity and efficient algorithms[J]. IEEE Transactions on Reliability, 1999, 48(1):87-95

[5] Yang M, Wang L N. Research of Web service reliability enhancement method based on trust fault tolerant[J]. Journal on Communications, 2010, 31(9):131-138

[6] Tian G H, Meng D, Zhan J F. Reliable Resource Provicion Policy for Cloud Computing[J]. Chinese Journal of Computers, 2010, 33(10):1859-1872

[7] Heath T, Martin R P, Nguyen T D. Improving cluster availability using workstation validation[A]// Proceedings of the ACM SIGMETRICS[C]. Marina Del Rey, California, USA, 2002:217-227

[8] Sahoo R K, Sivasubramaniam A, Squillante M S, et al. Failure data analysis of a large-scale heterogeneous server environment [A]// Proceedings of the DSN 2004[C]. Florence, Italy, 2004:772-784

[9] Shereshevsky M, Crowell J, Cukic B, et al. Software aging and multi-fractality of memory resources [A]// Proceedings of the 33rd DSN 2003[C]. San Francisco, Ca, USA, 2003:721-730

[10] Huang Y, Kintala C, Kolettis N, et al. Software rejuvenation: Analysis, module and applications[A]// Proceedings of the 25th Symposium on Fault Tolerant Computer Systems[C]. Pasadena, California, 1995:381-390

(上接第 123 页)

表 2 给出了 Coverity Prevent、Flawfinder 和本文中的方法测试漏洞的结果比较。从表 2 的测试结果来看，Coverity Prevent 的正确率较好，false_positives 的 3 个代码包的检测结果均保持在 96%左右，但存在一定的误报，Flawfinder 的结果略低。但 Encapsulation 类的漏洞的语法解析树的漏洞发现方法可以 100%发现漏洞，当然有一部分原因是在测试前已知此漏洞的存在，在契约规则的编写过程中会有一定的影响。由于特殊环境要求，例如代码包 B，能够发现在自定义传输数据包中发现的漏洞并归类到 Errors 类。由此可以得到 Coverity Prevent 和 Flawfinder 发现常规漏洞的效果明显较好，但在处理特殊环境和特殊参数情况下，类似自定义类和自定义传输协议数据包情况下，语法解析树的漏洞发现方法的契约规则更灵活，如果测试人员能够提前了解代码的功能，则对契约规则的设置帮助非常大。在实际工作中，对源代码漏洞检测可将不同的工具结合使用。另外虽然 Flawfinder 的漏洞发现要略低于 Coverity Prevent，但其由于属于开源软件，费用更低些。

结束语 针对源代码漏洞检测是一项相当复杂又繁琐的工作，需要程序员细心地了解每个调用函数或者对象方法的内部代码，利用改进的语法解析树解析函数调用前后形成的契约规则来判断函数是否存在漏洞，尤其改造了契约规则文法以支持对象之间存在继承而形成的契约规则。这种方法针对对象类继承关系下的漏洞检测和排查效果明显，并易于扩展；将其与已有的静态漏洞排查软件结合，以增加发现概率。

参 考 文 献

[1] Coverity[CP/OL]. http://www.coverity.com,2012

[2] CodeCheck[CP/OL]. http://www.abraxas-software.com/,2012

[3] FlawFinder HomePage[OL]. http://www.dwheeler.comlflawfinder/

[4] Bloch V J, Kohno J T T, McGraw G. ITS4: A Static Vulnerability Scanner for C and C++ Code[C]// Proc. 16th Computer Security Applications Conferences. New Orleans, LA, 2000:257-266

[5] Bauer T, Lips H P, Thiele G, et al. Operational tests on HVDC thyristor modules in a synthetic test circuit for the sylmar east restoration project[J]. IEEE Transactions on Power Delivery, 1997

[6] 张晓琳，王国仁. 用继承扩展 XML-RL[J]. 小型微型计算机系统，2005，26(2)：243-247

[7] 阳小奇，刘坚. 一种基于契约的跨过程安全分析方法[J]. 西安电子科技大学学报：自然科学版，2006，33(3)：390-394

[8] 陈海明，董韫美. 上下文无关语言分析树的一种表示形式[J]. 计算机研究与发展，2000，37(10)：1181-1184

[9] 陈再良，徐德智，陈学工，等. 基于链式结构 XML 文档的生成方法[J]. 计算机工程，2006，32(10)：59-61

[10] 肖袁. 一种高效的 XML 多分支路径查询算法[J]. 计算机应用与软件，2010，27(7)：153-155

[11] Swiler L P, Phillips C, Ellis D. Computer-attack Graph Generation Tool[C]// Proceedings of the 2nd DARPA Information Survivability Conference & Exposition. Los Alamitos, California, USA: IEEE Computer Society, 2001:307-321

[12] ISC. Internet Domain Survey[OL]. http://www.isc.org/ds/

[13] SCAP. Security Content Automation Protocol [OL]. http://scap.nist.gov/

[14] CVE. Common Vulnerabilities and Exposures. http://cve.mitre.org/

[15] CERT/CC. CERT/CC Statistics[OL]. http://www.cert.org/stats/

[16] 雷富兴，张来顺. 基于 Hoare 逻辑的过程调用的形式化方法[J]. 计算机工程与设计，2011，32(1)：197-201