软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(2):348−363 [doi: 10.13328/j.cnki.jos.004786]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于特征矩阵的软件脆弱性代码克隆检测方法*

甘水滔[1], 秦晓军[1], 陈左宁[1], 王林章[2]

[1](数学工程与先进计算国家重点实验室(无锡江南计算技术研究所),江苏 无锡 214083)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 甘水滔, E-mail: ganshuitao@gmail.com

摘 要: 提出了一种基于特征矩阵的软件代码克隆检测方法.在此基础上,实现了针对多类脆弱性的检测模型.基于对脆弱代码的语法和语义特征分析,从语法分析树抽取特定的关键节点类型描述不同的脆弱性类型,将 4 种基本克隆类型细化拓展到更多类,通过遍历代码片段对应的语法分析树中关键节点的数量,构造对应的特征矩阵.从公开漏洞数据库中抽取部分实例作为基本知识库,通过对代码进行基于多种克隆类型的聚类计算,达到了从被测软件代码中检测脆弱代码的目的.与基于单一特征向量的检测方法相比,对脆弱性特征的描述更加精确,更具有针对性,并且弥补了形式化检测方法在脆弱性类型覆盖能力上的不足.在对 android-kernel 代码的测试中发现了 9 个脆弱性.对不同规模软件代码的测试结果表明,该方法的时间开销和被测代码规模成线性关系.

关键词: 脆弱性检测;代码克隆;语法分析树;特征矩阵

中图法分类号: TP311

中文引用格式: 甘水滔,秦晓军,陈左宁,王林章.一种基于特征矩阵的软件脆弱性代码克隆检测方法.软件学报,2015,26(2):348−363. http://www.jos.org.cn/1000-9825/4786.htm

英文引用格式: Gan ST, Qin XJ, Chen ZN, Wang LZ. Software vulnerability code clone detection method based on characteristic metrics. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):348−363 (in Chinese). http://www.jos.org.cn/1000-9825/4786.htm

Software Vulnerability Code Clone Detection Method Based on Characteristic Metrics

GAN Shui-Tao[1], QIN Xiao-Jun[1], CHEN Zuo-Ning[1], WANG Lin-Zhang[2]

[1](State Key Laboratory of Mathematical Engineering and Advanced Computing (Jiangnan Institute of Computing Technique), Wuxi 214083, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: This article proposes a clone detection method based on a program characteristic metrics. Though analyzing the syntax and semantic characteristics of vulnerabilities, this detection method abstracts certain key nodes which describe different forms of vulnerability type from syntax parser tree, and expands four basic types of code clone to auxiliary classes. The characteristic metrics of the code then is finalized by obtaining the number of key nodes which are calculated via scanning corresponding code segment in the syntax parser tree. The clone detection based on a characteristic metrics creates basic knowledge base by extracting partial instances of open vulnerability database, and precisely locates the vulnerability codes by performing cluster calculation on the same codes responding to multiple types of code clone. Comparing with the detection method based on single characteristic vector, the proposed method produces more precise description about vulnerability. This detection method also offers a remedy to the drawbacks of formal detection method on its vulnerability type covering ability. Nine vulnerabilities are detected in an android-kernel system test. Testing on software of different code sizes shows that the performance of this method is linear with the size of the code.

Key words: vulnerabilitydetection; codeclone; syntax parser tree; metrics of characteristics

* 基金项目: 国家自然科学基金(91318301, 61170066, 6147179)

收稿时间: 2014-07-09; 修改时间: 2014-10-31; 定稿时间: 2014-11-26

部分软件代码脆弱性,如缓冲区溢出、内存多次释放、指针误用、竞争条件、资源泄漏、格式串溢出、整数溢出、原子性错误等,在语法和语义规则方面与程序设计语言的正确特性相违背[1,2],通过形式化的方法可以检测,但对于认证绕过、域保护不当、函数不恰当选项、后门代码等脆弱性类型,往往在代码形式上并不违背正确的语法和语义特征,常用的基于规则检查的方法难以识别[3−8].另一方面,代码复用在软件开发过程中极其常见,如果被复用的代码源中存在脆弱性,那么新建立的代码往往会延续同样的问题.对于上述两种情况,如果建立一个已发现的脆弱性代码数据库,从代码的相似度层面进行检测,查找某个软件系统出现的脆弱性代码在其他软件中的分布情况,就可以检测出更多的脆弱点,有效弥补静态分析方法在脆弱性检测类型和漏报方面的不足.

与基于规则检查的静态分析方法相比,基于克隆检测方法的脆弱性检测技术所具备的优势如下:

1) 克隆检测方法从被测代码与脆弱代码的相似度入手,能够检测无法用语法或语义规则描述的脆弱性.
2) 结合预定义脆弱性知识库,容易从词法级别对测试对象进行无关代码的过滤,降低了开销,可以应用于大规模代码的检测.
3) 通过脆弱性知识库的扩展,不仅可以进行脆弱性检测,而且提供了自动修补的可能性.

1 克隆检测方法现状及脆弱性检测应用分析

克隆代码是指软件系统中存在的相同或相似的代码片段.现代软件技术利用面向对象技术、设计模式、泛型编程、基于构件编程、面向方面编程(aspect oriented programming,简称 AOP)等强调高度的软件复用,同时也导致了脆弱性的扩散.由于缺乏追踪机制和手段,一旦被复用代码携带了某些脆弱性,一来难以得到及时的修补,存在严重的安全隐患;二来即使得到修补,也可能要付出过大的代价.面向源代码的代码克隆检测方法通过建立特定的脆弱性代码数据库,对相关软件源代码进行克隆检测分析,挖掘软件中与脆弱性代码库相关的克隆代码,能够快速和有效地定位高风险脆弱性.

代码克隆有多种类型,被普遍认可的是 Roy 等人的分类[9].这种分类方法从程序代码的词法和语义上把克隆代码的基本类型归纳为如下 4 种:

1) 类型 1:两克隆代码片段之间除了空白和注释有差异外,其他部分完全相同.
2) 类型 2:两克隆代码片段之间除了空白、注释、变量名、变量类型及常量值等词法特征有所差异外,其他部分的语法结构完全相同.
3) 类型 3:两克隆代码片段之间除了类型 2 中的差异外,还有少量的语句增减及修改.
4) 类型 4:两克隆代码片段语义相同,但语法结构不同.

通过对 NVD,CVE 等公开漏洞库的调查,可以发现不同的软件系统中出现大量由于代码克隆引入的脆弱性,原因来自于代码复制、软件系统在同一框架下进行开发,或者围绕某个软件系统进行的第三方软件开发.具体可分为 3 种情况:

1) 不同的软件系统之间使用了相同的代码库,主要在代码结构、函数名称、参数变量、常量、操作符方面有高度的相似度;
2) 不同的软件系统之间使用了相同的函数库或 API;
3) 不同的软件系统之间使用了相同的算法、协议或产品说明书.

从 2010 年开始,出现了克隆检测技术在软件脆弱性挖掘领域的应用研究,爱荷华州立大学的 Pham 等人提出的 SecureSync 原型系统[10,11]设计了 xASTs 和 xGRUMs 两种模型:xASTs 利用抽象语法树描述代码库复用引发的脆弱性,xGRUMs 利用有向图描述 API 复用引发的脆弱性.这两个模型利用 Exas 的特征向量提取方法,统计关键节点和节点组合的出现次数构造特征向量.SecureSync 通过整理的 119 个开源软件系统(176 个版本)中发布的 60 个漏洞用以检测模型性能,定位了多个软件系统中的脆弱性.Li 等人在 2011 年开发 CBCD 原型系统[12],利用程序依赖图解析脆弱性代码和被测代码信息,将系统依赖图划分成多个节点数量较少的子图,提取其中脆

弱性敏感的函数调用节点,通过图的同构匹配算法查找近似脆弱性的代码.CBCD 收集了来自多个开源程序的53 个公开漏洞代码,在对 Linux 下的多个开源应用程序的测试中准确定位了其中的 20 个脆弱性.

由国内外研究文献可知,目前克隆检测方法在脆弱性检测领域的应用较少,只出现了 SecureSync 和 CBCD 两个原型系统,从中可以发现其中存在的诸多难点:

1) 准确的脆弱性量化特征提取.在代码相似性度量层面,需要对代码特征进行量化,即,把软件转化为各种中间代码表示形式,在其上进行相关词法或语义的量化和抽取.针对不同类别的脆弱性,需要对脆弱性的机制原理、程序设计语言相关特性进行高度的特征抽象,而且该特征集合必须能够较好地区分脆弱性代码和正确代码,以免后期测试中出现大量的虚报.因此,量化特征的提取需要针对已发现的脆弱性做大量针对性的工作.准确而又能够覆盖多种类别脆弱性的特征提取,是该技术应用的主要难点之一.
2) 脆弱代码知识库的构建.脆弱代码知识库是克隆检测方法的基础,库内容的详细程度对检测效果有决定性的影响,而且需要根据检测方法的不同,对库中每个漏洞建立诸如词法特性、语法特性等多种条目,工作量巨大.另一方面,公开漏洞库中的漏洞描述信息一般过于模糊,多数情况下需要结合补丁比较定位具体代码,还需要过滤冗余代码,提取最具脆弱特征的相关代码,这需要对各种脆弱性类型有深入准确的机制原理分析.此外,还需要对克隆源和被测目标软件代码使用的开发环境进行分析,找出开发过程所使用的代码库、函数库、功能模块中可能存在的脆弱性.
3) 代码切分.脆弱性相关代码的复用在软件系统的开发过程中普遍存在,但大部分复用的是修补过的正常代码[13].由于脆弱代码和修补代码之间存在的差异大多是微弱的,可能造成过高的虚报率.克隆检测中,代码切分的完备性决定了漏报的可能性,一般树和图两类克隆检测通过划分成子树和子图进行代码分片,但是代码分片的细粒度越小,检测的时间复杂度越高,需要合理地进行代码切分以控制脆弱性漏报的程度.
4) 脆弱性类型和克隆代码类型对克隆代码分析的影响.SecureSync 和 CBCD 两个系统都既不区分脆弱性类型,也不区分克隆代码类型.在脆弱性特征提取上采用统一的标准,会弱化不同类型脆弱性的差异,引发虚报或漏报.
5) 中间代码的选择.必须将测试对象转换成易于理解和遍历的中间代码形式,脆弱性特征的描述也是在中间代码上进行的,好的中间代码设计也是该技术应用的关键点和难点之一.

为了克服以上提出几个难点,本文提出了一种基于特征矩阵的代码克隆检测方法的多类脆弱性检测模型 CVdetector,在脆弱性量化特征提取、脆弱性知识库的构建和中间代码的选择等重要技术环节中采用了大量先验和后验信息,完成了原型系统构建.在对 android-kernel,dnsmasq 等开源代码的测试过程中,准确地发现了其中的多个脆弱点,表现出较好的脆弱性检测能力和运行效率.其基本思路如下:

1) 考虑了脆弱性特征和代码克隆类型对检测结果的影响.本文基于对脆弱代码的语法和语义特征分析,将 4 种基本克隆类型细化拓展到更多类,利用语法分析树提取不同类型的克隆代码特征构造被测代码的特征矩阵,其中的每个特征向量表示一种克隆类型对应的特征;通过分析脆弱性特征与代码克隆类型的对应关系构造脆弱代码的特征矩阵,最后对脆弱代码和被测代码的特征矩阵进行聚类操作,完成更高精度的脆弱性检测.
2) 采用一种半自动化方法采集公开漏洞代码,利用 Google code search[14]中提供的大量开源项目版本和代码资源,一方面人工从 CVE、NVD、绿盟的公开漏洞报告信息获取大概漏洞代码信息(版本、关键函数名称或文件位置等);另一方面,基于 Google code search 提供的正则表达式实现对不同版本中漏洞相关代码的比较,提取准确的脆弱代码.
3) 综合程序依赖图和抽象语法树两种代码表示结构的优势,分别应用于代码克隆检测的不同环节.利用基于程序依赖图的切片分析对被测代码进行初步的代码子集划分,每个代码片段的语句都成数据依赖和控制依赖关系.由于图的结构不便于遍历操作,本文在特征提取环节采用树的结构进行遍历,

通过限定树的节点数目做进一步的代码划分.

与已有工作相比,基于特征矩阵的检测方法对脆弱代码具有更好的针对性,结果更为精确.

2 基于特征矩阵的代码克隆检测方法的多类脆弱性检测模型

2.1 CVdetector模型框架

CVdetector 整体框架如图 1 所示,主要研究内容包括以下 9 个模块:多类脆弱性量化特征分析、建立脆弱性数据库、代码切片、不同脆弱性的克隆引入类型分析、语法分析树转换、不同克隆代码类型的语法分析树特征提取、特征向量和特征矩阵的生成、无关代码过滤、脆弱代码和被测代码的聚类.

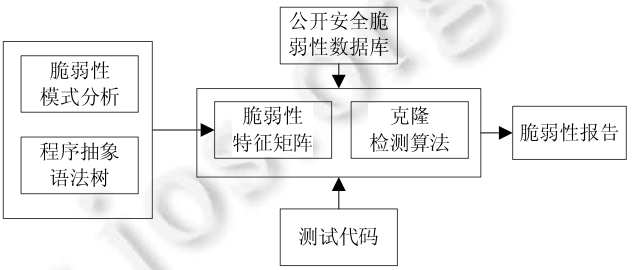

Fig.1 CVdetector's framework

图 1 CVdetector 模型框架

(1) 多类脆弱性量化特征分析

SecureSync 系统对 US-CERT 中的多个脆弱性报告人工分类分析,在 Firefox,Thunderbird,SeaMonkey,Tomcat 等软件系统中发现了大量克隆脆弱性实例.本文结合人工分析,筛选了 CVE 数据库中的 980 个漏洞报告,利用词法分析技术分成相关的 100 个子集,发现其中的 18 个子集中的 42 个脆弱性是克隆代码引发的脆弱性.CVdetector 模型通过对不同类别的脆弱性机制原理分析,针对格式串溢出脆弱性、安全系统调用、整数溢出、竞争条件、资源保护不当、后门代码、资源泄露、函数不恰当参数、认证缺失等类型脆弱性,分析各自在克隆代码过程中的变化情况,结合对应代码的语法分析树,提炼可以表达各种脆弱性类型信息的节点或节点结构信息,以区别其他代码,有效降低虚报率.

(2) 建立脆弱性数据库

针对特定的测试目标软件,选择功能相关、开发框架相关的软件系统作为克隆源,基于漏洞数据库提取克隆源中与脆弱性相关的代码特征,作为脆弱性挖掘的知识库.脆弱性知识库是整个 CVdetector 模型脆弱性挖掘能力的基础,知识库中脆弱性类别和各类脆弱性条目的数量决定了对测试目标未知脆弱性挖掘能力的差异.

(3) 代码切片

该环节主要是对当前克隆检测方法在脆弱性检测领域应用不足提出的优化.目前主流的克隆检测方法中,基于图的检测方法能够准确定位与控制流信息和数据流信息相关的代码,可解决不连续代码的检测问题,但图遍历的复杂度制约该方法在大规模代码上的应用.树结构便于遍历,基于树的克隆检测方法可以实现测试的完备性,但只能实现对连续代码的测试.基于度量值的检测方法可以实现快速测试,适用于大规模代码目标软件,但对代码的量化特征提取是实现克隆检测的关键.针对以上方法的优缺点,CVdetector 模型首先利用代码切片方法(基于图)将测试目标软件中功能相关的非连续代码提取组合成连续代码,把目标软件切分成多个连续代码块,然后使用语法分析树和量化特征度量实现对连续代码块的测试.若实现上,该步骤使用 codesurfer 工具[15]对程序依赖图进行静态切片[16],提取切片后程序.

(4) 不同脆弱性的克隆引入类型分析

针对筛选的公开漏洞库中的脆弱性代码,需要对每个脆弱性条目进行类型划分,结合脆弱性机制原理分析,

梳理各个条目可能引发的克隆代码形式,以确定对应的特征矩阵提取.如图 2 所示.

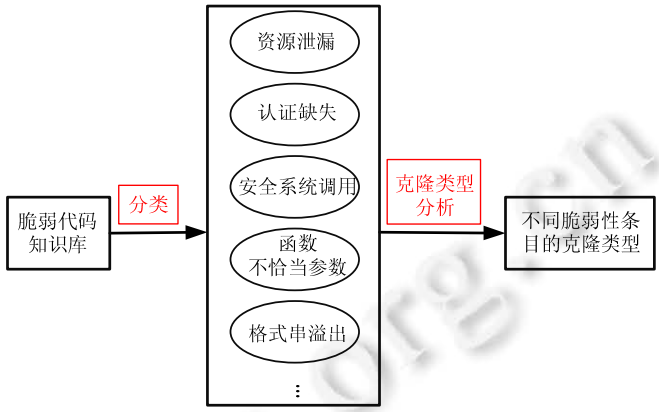

Fig.2 Clone type analysis of vulnerability

图 2 脆弱性的克隆类型分析

(5) 语法分析树转换

针对上一个环节抽取的代码片段,实现代码到语法分析树的映射,语法分析树采用树的形式表达源代码语法、语义结构逻辑信息.其中,叶子节点代表源代码某个词法单元,中间节点的类型可能是某个变量类型、声明语句、赋值语句、算术运算语句、判断语句、循环语句或函数实现过程等,一棵子树表示一段连续的源代码,每段代码被解析成由多种类型的节点构成的语法分析树.

(6) 不同克隆代码类型的语法分析树特征提取

特征提取是整个模型的核心环节,需要理解语法分析树的各种节点类型,通过分析不同形式的克隆代码和原始代码的词法、语法差异,提取表达代码相似性的相关节点,忽略体现代码差异的相关节点.每个代码片段可能有多种克隆形式,需要进行多种语法节点的提取,形成不同克隆代码类型的特征集合.

(7) 特征向量和特征矩阵的生成

模型通过统计各相关节点在语法分析树中出现的次数,以构造相应的特征矩阵.对于脆弱性知识库中的脆弱代码和待测的目标代码片段,需要分别生成预定义的各克隆类型所对应的不同特征向量以构成特征矩阵.特征矩阵是对特征向量的扩展,只需对语法分析树进行一次遍历.

(8) 无关代码过滤

针对庞大的被测源代码集合,在进行聚类分析之前需要过滤无关的代码片段,以降低空间和时间开销.本文采用两种过滤方法:第 1 种是基于文本的过滤方法,利用词法分析技术从脆弱代码库中提取脆弱性关键字,然后扫描测试源代码,过滤与脆弱性关键字无关的部分;第 2 种是基于结构的过滤方法,计算脆弱代码片段的特征向量和被测代码片段的特征向量的相关度,过滤相关度较低的测试片段.

(9) 脆弱代码和被测代码的聚类

经过前几个步骤,每一个脆弱代码和被测代码片段都生成了一个特征矩阵,本步骤对两个特征矩阵集合中的特征向量进行聚类运算,以发现可能存在的克隆对,检测脆弱性是否存在.

2.2 不同克隆代码类型的语法分析树特征提取

程序代码的语法分析树的叶子节点用代码的词法单元表示.图 3 是表 1 中的一段代码简化过的语法分析树及特征向量生成表示的示例.

Table 1 Syntax analysis tree's key node information of 4 clone code forms

表 1 4 种克隆代码形式的语法分析树关键节点信息

节点类型编号	节点命名	节点表示信息	克隆代码类型 1	克隆代码类型 2	克隆代码类型 3	克隆代码类型 4
1	for	for 循环结构	√	√	√	×
2	stmtexp	一条语句	√	√	√	√
3	decl	声明操作	√	√	√	√
4	incr	自加操作	√	√	√	√
5	cond	比较操作	√	√	√	√
6	vari	普通变量	√	√	×	×
7	para	环境变量	√	√	×	×
8	assign	赋值操作	√	√	√	√
9	block	大括弧包含的程序	√	√	√	×
10	mul	乘法操作	√	√	√	√
11	add	加法操作	√	√	√	√
12	cons	常量	√	√	×	×
13	type	变量类型	√	×	√	√
14	fun_call	函数调用	√	√	√	√
15	fname	函数名称	√	√	√	√
16	fpara	函数参数	√	√	√	√

注: √ 表示关键节点,× 表示非关键节点

一段原始代码:

```
for (int i=1; i<data; i++){
    cont=cont+i;
    dod=dod*i;
    search(cont,dod);
}
```

克隆类型 1:

```
for (int i=1; i<data; i++){
    cont=cont+i;
    dod=dod*i;
    search(cont,dod);
/* xx */
}
```

克隆类型 2:

```
for (u_int k=1; k<data1; k++){
    cont=cont+i;
    dod=dod*i;
    search(dod,cont);
}
```

克隆类型 3:

```
for (int i=1; i<data; i++){
    cont=cont+i;
    dod=dod*i;
    search(cont,dod,data);
}
```

克隆类型 4:

```
while (i<data){
    cont=cont+i;
    dod=dod*i;
    search(cont,dod,dat
a);
    i++;
}
```

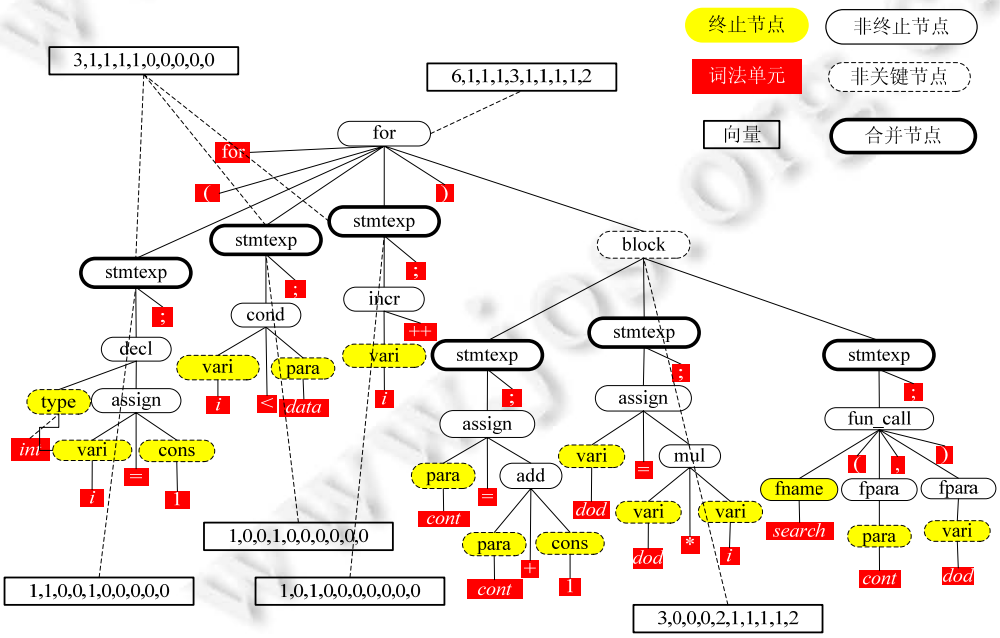

Fig.3 Syntax analysis tree of a simplified code fragment and characteristic vector generation representation

图 3 一段代码简化过的语法分析树及特征向量生成表示

节点属性包括以下内容:

1) 相关节点:一棵简化的语法分析树的所有类型节点,即,从完全的语法分析树节点中抽取出与所需代码特征相关的节点,排除无关节点,构成简化的语法分析树.图 3 中的相关节点包括表 1 中的 16 种节点,不相关节点不列入特征向量计算,图中没有显示.相关节点的不同属性组合成特征向量的不同元素,相关节点可分为关键节点和非关键节点.
2) 关键节点:表征需研究的代码片段特征的节点,必须生成相对应的统计特征向量或矩阵,并且要作为克隆检测的主要计算对象.图 3 中的关键节点包括 stmtexp,decl,incr,cond,assign,mul,add,fun_call,fname 和 fpara.
3) 非关键节点:不影响克隆检测结果,但在生成关键节点特征向量或矩阵的过程中可能会使用到的节点.图 3 中的非关键节点包括 for,vari,para,cons,type,block.非关键节点作为子树的根节点时,需要统计特征向量,但自身不具备统计效应.
4) 终止节点:只连接一个词法单元的节点,可以是关键节点,也可以是非关键节点.图 3 中,type,vari,cons,para 是终止节点,也是非关键节点,在特征向量提取过程中忽略.
5) 合并节点:需要和相邻的兄弟节点特征矩阵合并的节点.合并节点本身无含义,仅为便于计算而设立.

特征矩阵相关的处理步骤主要包括以下 3 步:

(1) 特征向量生成

特征向量按所属的不同克隆类型从语法分析树提取,需要提取所有相关节点的信息.对于表 1 中的原始代码,其中的 for 循环代码具有 4 种克隆形式,表 1 中结合代码特点抽取了语法分析树中的 16 种节点类型作为相关节点,每段连续的代码(子树)的特征向量由 16 维向量构成.即,(for,stmtexp,decl,incr,cond,vari,para,assign,block,mul,add,cons,type,fun_call,fname,fpara).根据不同类型的克隆代码和原始代码差异,定义了各自的关键节点,使得原始代码和克隆代码之间的语法和语义特征能够充分体现.图 3 中的语法分析树对应第 4 种克隆代码类型,其中定义了 6 个非关键节点:(for,vari,para,cons,type 和 block).第 4 种克隆形式改变了整个语法结构,需要通过定义 for 节点、block 节点及 which 节点(因篇幅关系,表中省略了该节点描述)为非关键节点来隐藏不同的循环结构差异;代码中出现了某些参数、变量的增添和删减操作,因此将 vari,para 和 cons 节点定义为非关键节点来隐去参数和变量带来的代码差异.为清晰起见,图中描述的特征向量忽略了 6 个非关键节点,用 10 维向量(stmtexp,decl,incr,cond,assign,mul,add,fun_call,fname,fpara)来描述.

语法分析树中的节点可能有两种作用:一种是作为单个节点出现,只代表节点本身;另一种是作为子树的根节点出现,代表整个子树.

- 当作为单个节点时,需要初始化特征向量:非关键节点用零向量初始化,关键节点用对应的单位向量初始化.例如,vari 节点初始化为(0,0,0,0,0,0,0,0,0,0),assign 节点初始化为(0,0,0,0,1,0,0,0,0,0)等;
- 当树中的节点描述为某子树的根节点时,其特征向量为其对应的不同类型关键子节点出现的频数和,如图 3 中 for 节点代表的子树,特征向量计算为(6,1,1,1,3,1,1,1,1,2),该特征向量为孩子节点特征向量的累加,即,for(0,0,0,0,0,0,0,0,0,0),stmexp(1,1,0,0,1,0,0,0,0,0),stmexp(1,0,0,1,0,0,0,0,0,0),stmexp(1,0,1,0,0,0,0,0,0,0),block(3,0,0,0,2,1,1,1,1,2)等节点特征向量的累加和.

(2) 特征矩阵生成

特征矩阵的生成是对特征向量生成的一个扩展,表 1 中的特征矩阵生成是 16×1 矩阵到 16×4 矩阵的扩展操作,每个特征矩阵的行向量对应一种克隆代码形式的关键节点向量,非关键节点默认为 0.图 4 是对 for 根节点的特征矩阵的生成描述,其特征矩阵为所有子节点特征矩阵和 for 节点初始化特征矩阵的累加和.模型需要对整棵树进行一次后序遍历操作生成某些节点的特征矩阵.

(3) 特征矩阵合并

单个根节点的特征矩阵只能映射到部分的连续代码片段,大部分连续代码片段的特征矩阵需要通过对相邻节点进行合并操作生成.合并操作需要设置参数控制合并节点的数量,该参数与节点的词法单元总数、合并节点数量相关,选取的原则是尽可能地降低误报率.在图 4 中,特征矩阵是合并了 stmtexp 等 3 个相邻的兄弟节点

特征矩阵,可映射到“*int i*=1; *i*<*data*; *i*++”这段连续的代码.

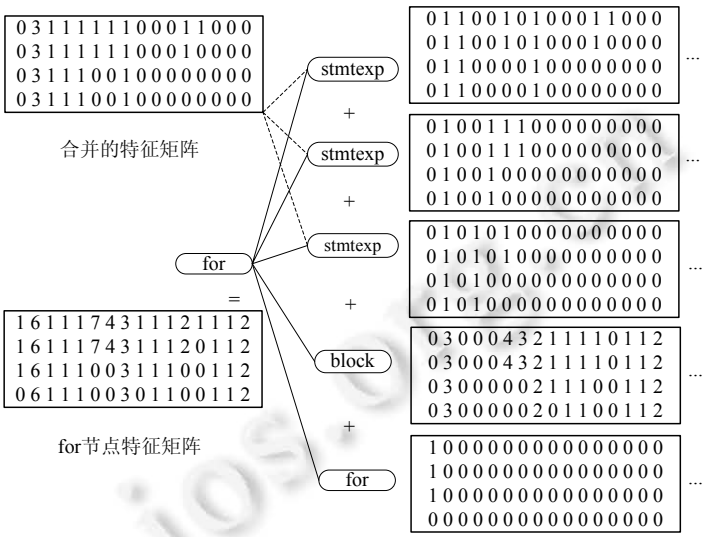

Fig.4 Characteristic matrices generation of “for” node

图 4 for 节点的特征矩阵生成

2.3 关键算法描述

定义 1(代码特征矩阵). 一棵由 k 种节点类型的语法分析树构成了一张符号表 $\psi=\{\tau_1,\tau_2,\ldots,\tau_k\}$,$k$ 为所有节点类型数量,程序代码的特征符号矩阵(以符号代替真实数值)用 $SymM=[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}$ 描述.对于 $SymM$ 中的任意元素 $SymM[i][j]$,满足 $SymM[i][j]\in\psi$.一段具体代码 C 对应的特征矩阵用 M 表示: $M=[v_1,v_2,...,v_{n-1},v_n]^T_{n\times m}$, n 为使用到的代码克隆类型数量.其中,v_i 为第 i 个 m 维特征向量,描述为 $v_i=(x_{i1},x_{i2},\ldots,x_{i(m-1)},x_{im})$,表示第 i 种克隆类型对应的特征向量,其中,x_{ij} 表示第 j 种语法节点或多个节点的组合出现的次数,m 为选择的相关节点类型数量.

定义 2(q-级原子树模式和 q-级特征向量)[17]. 一个 q-级原子树式是一棵高度为 q 的完全二叉树.对于给定的标签集 L,其中包括空标签 ε,最多包含 $|L|^{2^q-1}$ 个不同的模式.

对于树 T,对应的 q-级特征向量用 $v_q(T)=\langle b_1,b_2,...,b_{|L|^{2^q-1}}\rangle$ 表示,其中,b_i 表示为树 T 中第 i 个 q-级原子模式的出现次数(注:本模型为 1-级特征向量).

定义 3(偏序关系). 对于特征符号矩阵 $SymM=[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}$ 中任意两个符号向量 v'_i,v'_j, 如果满足:对于 $\forall x\in v'_i,\exists y\in v'_j$, 使得 $y=x$,那么 $v'_i\leqslant v'_j$. 偏序关系具有传递性.

定义 4(向量距离度量). 对于 m 维向量 $v_1=\{x_{11},x_{12},\ldots,x_{1(m-1)},x_{1m}\}$ 和 $v_2=\{x_{21},x_{22},\ldots,x_{2(m-1)},x_{2m}\}$,两向量汉明距离为 $D_H(v_1,v_2)=\|v_1-v_2\|_1=\sum_{i=1}^{m}|x_{1i}-x_{2i}|$,两向量的欧式距离为 $D_O(v_1,v_2)=\|v_1-v_2\|_2=\sqrt{\sum_{i=1}^{m}(x_{1i}-x_{2i})^2}$.

定义 5(偏序关系和向量距离相关关系). 对于特征符号矩阵 $SymM=[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}$ 中的向量 v'_i 和 v'_j,若存在偏序关系 $v'_i\leqslant v'_j$,那么对于代码片段 C 和 C',符号向量 v'_i 和 v'_j 所对应的代码 C 中的特征向量分别为 v_i 和 v_j,所对应代码 C'中的特征向量分别为 u_i 和 u_j,满足向量距离 $D(v_i,u_i)\leqslant D(v_j,u_j)$.其中,$D$ 是任意距离度量.

定义 6(s-类克隆对关系). 对于代码片段集合 $SetC=\{c_1,c_2,\ldots,c_k\}$,代码的特征符号矩阵:

$$SymM=[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}.$$

代码片段集合 $SetC$ 对应的特征矩阵集合为 $SetM=\{M_1,M_2,\ldots,M_k\}$,对于任意的 $M_i,M_j\in SetM$,如果存在第 s($1\leqslant s\leqslant n$)维特征向量 $M_i[s],M_j[s]$,满足 $D(M_i[s],M_j[s])\leqslant\sigma$,$\sigma$为人工设定的阈值,那么代码片段 c_i 和 c_j 成 s-类克隆对

关系.

定义 7(脆弱代码报告). 对于被测代码片段 c 和脆弱代码片段 c'成克隆对关系,代码 fc'为脆弱代码 c'修补过的正常代码,如果满足 $D(c,c')\leqslant D(c,fc')$,那么代码片段 c 为克隆 c'后脆弱代码.

本模型的整个脆弱代码检测过程包括多个关键算法和设计细节,以下对从其他文献中引用的算法环节仅进行标注,只对本文提出的关键算法进行较为详细的描述.

(1) 脆弱代码知识库的形成

本模型主要以 CVE 数据库和绿盟安全漏洞数据库为参考数据源,结合 Google code search 分析脆弱性代码和补丁代码信息,针对 Linuxkernel,dhcp,opensl,ntp,net-snmp,bind 这 6 个开源软件系统,分别定位了 96 条、11 条、8 条、4 条、2 条、10 条脆弱性代码,包括了格式串溢出脆弱性、安全系统调用、整数溢出、竞争条件、资源保护不当、后门代码、资源泄露、函数不恰当参数和认证缺失共 9 种脆弱性类型.对每个脆弱性代码片段,存储代码所在的整个函数,并通过人工分析方式过滤其中的无关代码,对脆弱性代码和补丁代码进行词法标注,便于后续克隆检测算法的识别.

(2) 代码切片

利用前向程序切片方法,基于不同的程序变量,从被测代码中提取与该变量有控制依赖关系和数据依赖关系的代码片段[18].由于变量间可能存在的相关性,这些代码片段之间有可能存在包含关系,再引用语义线程[19]的思想对代码片段进行合并,从被测代码中切分出一个没有冗余代码片段的最小划分集合.

(3) 语法分析树节点筛选

本模型使用 EDG Front-end Parser 将每个源代码文件解析成由 225 种节点类型构成的语法分析树,根据需要选择了其中的 100 种作为相关节点类型.

关键节点类型的选择不仅与基本克隆类型有关,而且需要考虑待检测脆弱性的代码特征.以 CVE-2009-0021 和 CVE-2009-0047 为例,这两个脆弱性分别出现在 NTP 和 Gale 中,都是因为没有对 EVP 库中 EVP_VerifyFinal 函数的返回值进行验证而引发了远程的安全认证绕过脆弱性.但这两个脆弱性的函数参数完全不一致,为描述这两段代码的相似性,关键节点类型中不应该包括表示参数类型、参数数量和参数名称等节点类型;CVE-2008-5023 漏洞同时出现在 Firefox 和 SeaMonkey 中,脆弱性相关的代码片段只有调用 CanExecuteScripts 函数的第 2 个参数名称不同,因而从语法分析树中选择关键节点类型时不应该包括表示参数名称相关的节点类型;BUGTRAQ id:36859 空指针脆弱性出现在 OpenBSD 中的 ip_ctloutput 函数和 ip6_ctloutput 函数中,这两个函数功能一样但名称不同;而 CVE-2011-2484 为整数类型转换引发的脆弱性,出现在 openbsd 和 linux kernel 中,只是 drm_modeset_ctl 函数中的代码结构和某些变量类型存在差异.因此,选择关键节点类型时,应针对不同的脆弱性类型或代码片段分析该脆弱性可能呈现的各种克隆代码特征,以更加准确地描述 4 种基本克隆代码类型中任意一种类型呈现的多种形式.

为了更充分地体现脆弱性代码的语法和语义特征,提高检测的针对性,本文通过对知识库中 9 种脆弱性类型的分析,将 4 种基本克隆类型进一步细化为 10 种,分别定义其中的关键节点.其中,最少的克隆类型包含 56 种关键节点,最多的有 96 种,总相关节点种类为 100.实现上,把每个代码片段映射成 100×10 的特征矩阵.

(4) 特征矩阵生成

代码片段的特征矩阵包括脆弱代码片段和被测代码片段的特征矩阵.对脆弱代码,基于脆弱代码知识库将一个脆弱性条目切分成包含不同词法单元的多个脆弱代码片段.由于利用自动遍历计算其子树映射的脆弱代码子集可能会产生多个不包含核心脆弱代码语句的代码片段,导致过高的虚报率,因此,本模型利用人工切片分析,确保切分出的脆弱代码片段都必须包含最核心的脆弱代码语句.完成切分后,生成每个脆弱代码片段 M'的单个特征矩阵,同时记录相应的词法单元数量 M' *Tokens*,最后将脆弱代码片段中最大词法单元记为 *MaxTokens*,最小词法单元数量记为 *MinTokens*.对被测代码集合,采用后缀方式对代码片段的语法分析树进行遍历,选择词法单元数量在[*MinTokens*,*MaxTokens*]范围中的子树生成相应的特征矩阵.算法 1 描述了被测代码的特征矩阵遍历生成过程.

算法 1. 特征矩阵遍历生成算法 *CharacterMatrixGenerate*.

输入:

T:代码片段对应的语法树;

SymM:特征符号矩阵 $[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}$.

输出:

SetM:特征矩阵集合.

1: $k\leftarrow 0$, $SetM\leftarrow\{\varnothing\}$

//把各种类型的相关节点映射到具体的整数索引值

2: **foreach** relevant node

3: $id.node\leftarrow k$

4: $k\leftarrow k+1$

5: **endfor**

//初始化关键节点个体的特征矩阵 $M0_{node}$

6: **foreach**significantnode

7: **foreach** vecter v'_i **in** *SymM*

8: **if** node is significant **in** v'_i

9: $M0_{node}[i][id.node]\leftarrow 1$

10: **endif**

11: **endfor**

12: **endfor**

//对树 *T* 进行节点后缀遍历

13: **foreach** node **in** *T* traversedin post-order

14: $M_{node}\leftarrow\sum M_{chindren(node)}+M_{node}$

15: **if** node is significant

16: $M_{node}\leftarrow M_{node}+M0_{node}$

17: **endif**

//树的词法单元数量脆弱代码的词法单元数量范围中

18: **if** $MinTokens\leqslant T_{node}number\leqslant MaxTokens$

19: $SetM\leftarrow SetM\cup\{M_{node}\}$

20: **endif**

21: **endfor**

22: **return** *SetM*

(5) 特征矩阵合并和分组

由于被测代码的单个节点对应的特征矩阵集合不能覆盖大部分代码的子集,需要对某些相邻的兄弟节点进行合并操作以扩大特征矩阵集合.本模型使用的树的兄弟节点合并算法与 Deckard[17]的节点合并算法有所差异,不使用一个固定长度的滑动窗口控制需合并的相邻节点的数量,而是对词法单元总数处于[*MinTokens*, *MaxTokens*]范围中的相邻节点进行合并.特征矩阵分组是把词法数目相近的脆弱代码片段和被测代码片段划分到一个组中,分别对各个组进行克隆对检测,防止词法偏差较大的代码片段被误报为克隆对.

(6) 偏序关系分组

根据 10 种克隆类型定义的符号向量,需要对 10 个符号向量进行偏序关系分组,目的是为了减少克隆对检测的冗余操作.对于一个偏序关系组的两个符号向量 v'_i,v'_j,如果 $v'_i\geqslant v'_j$,针对两个代码片段 c,c',如果 c 和 c'成 i 类克隆对关系,那么 c 和 c'也成 j 类克隆对关系.偏序关系分组可以为代码片段的克隆对进行分组检测,从而减少

重复操作.偏序关系分组算法如下:

算法 2. 偏序关系分组算法 *CharacterVecterGroup*.

输入:

SymM:特征符号矩阵 $[v'_1,v'_2,...,v'_{n-1},v'_n]^T_{n\times m}$.

输出:

G:存在偏序关系的符号向量分组集合.

```
1:  k←0, G←{∅}
2:  foreach  v'_i ∈ SymM
3:      G_k ← {v'_i}
4:      foreach  v'_j ∈ SymM  and j≠i
5:          if  v'_i ≤ v'_j  or  v'_i ≥ v'_j
6:              G_k ← G_k ∪ v'_j
7:              SymM ← SymM − v'_j
8:          endif
9:      endfor
10:         G←G∪G_k
11:         k←k+1
12:     if SymM=∅
13:         return G
14:     endif
15: endfor
```

(7) 不同类克隆对检测

对于 n 个向量,传统的聚类算法需要对聚类对象做两两空间距离的计算,复杂度为 $o(n^2)$,对往往高达百万量级的 n,开销巨大.本模型采用基于局部敏感哈希(local sensitive Hash,简称 LSH)的聚类方法[20],只需对向量做一次哈希运算,聚类操作的时间开销接近线性.局部敏感哈希函数是 LSH 聚类方法的关键环节,对两个向量,该函数产生的哈希代码的相似概率和两向量的距离成严格递减的关系,两向量距离越小,哈希代码相同的概率就越大.利用局部敏感哈希函数可以把向量对应的代码映射到某个整数区域(也可以是某个具体的整数)中,代码片段的相似程度越高,映射到相同区域的概率就越大.

本文利用曼哈顿距离度量向量的相似性,如果$\|u-v\|=1$,则概率计算公式为

$$\Pr(l)=\Pr[h(u)=h(v)]=\int_0^w \frac{2.\mathrm{e}^{-\left(\frac{x}{l}\right)^2}}{\left(l.\sqrt{2\pi}\right)\left(1-\frac{x}{w}\right)}\mathrm{d}x,$$

其中, $h(u)=\dfrac{a.u+b}{w}$. 在该公式中,a 为向量,w 为固定的正实数,b 为[0,w]中的某个随机数.

$\Pr(1)$为关于 1 的递减函数,对于 $1\leqslant\delta$,满足 $\Pr(1)\geqslant p=\Pr(\delta)$.对于两个向量 u,v,满足:当$\|u-v\|\leqslant\delta$时,$\Pr[h(u)=h(v)]\geqslant p$.也就是说,若两向量距离在$\delta$范围内,则至少以 p 的概率映射到同一个整数上;若两向量距离在δ范围外时,则最多以 p 的概率映射到同一个整数上.为了减少误差,将采用一族独立的哈希函数:

$$H:h=(h_1,h_2,\ldots,h_k).$$

每个向量 u 在哈希函数族的映射下产生一个整数向量$(h_1(u),h_2(u),\ldots,h_k(u))$,最后生成的数值为

$$h(u)=\left[\sum_{i=1}^{k} r_i.h_i(u)\right] \bmod P,$$

其中,r_i 为随机选择的整数,P 为一个较大的素数.这样,若两向量距离在δ范围以内,则至少以 p^k 的概率映射到同

一个整数上;若两向量距离在δ范围以外,则以最多 p^k 的概率映射到同一个整数上.本模型采用 N 个独立的 k 哈希函数族,每个向量将被映射到 N 的独立的数据域中.如果原本符合克隆关系的向量 u 和 v 在某个哈希函数族中被漏报了,那么在其他的函数族中仍存在被报告的机会.两向量被 N 个哈希函数族漏报的概率不超过 $1-p^k$,如果 N 足够大,则这个概率几乎为 0.为了减小不同类克隆对间的相互干扰,本模型随机挑选多个代码片段,通过对各自特征矩阵的不同向量进行 Hash 聚类,学习出合适的δ阈值,使得绝大部分特征矩阵中的特征向量体现出差异.算法 3 和算法 4 给出了特征矩阵组的克隆对检测算法.

算法 3. 利用 LSH 聚类进行特征矩阵相似性检测算法 *CharacterMatrixCloneDetectionBasedOnLSH*.

输入:

SetM:被测代码的特征矩阵集合;

SetM′:脆弱代码的特征矩阵集合.

输出:

RG:不同脆弱代码片段的克隆对报告.

```
1:  RG←{∅}
2:  k←1
3:  //选择脆弱代码的特征矩阵
    foreach M′ in SetM′
4:      foreach M in SetM
5:          group_k←MLSH(SetM,M′,δ,j)
6:          if |group_k|≥1
7:              RG←RG∪{group_k}
8:          endif
9:              k←k+1
10:     endfor
11: endfor
12: return RG
```

算法 4. 脆弱代码的特征矩阵集合 *SetM′*中的特征矩阵.

输入:

M′:脆弱代码的特征矩阵集合 *SetM′*中的特征矩阵;

SetM:被测代码的特征矩阵集合;

δ:相似度度量阈值;

G:存在偏序关系的符号向量分组集合.

输出:

返回的与脆弱代码片段 *M′*成克隆对关系的被测代码片段集合.

```
1:  group←{∅}
2:  foreach M in SetM
//M.index 用来存储被测代码片段 M 和 M′成克隆对关系时的符号向量类别集合
3:      M.index←{∅}
4:  endfor
5:  foreach G_i in G
//按照偏序大小关系选择
6:  foreach  v′_j  in G_i
7:      SetM_1←SetM
```

8: **while** $SetM_1 \neq \varnothing$ **do**
9: **foreach** M **in** $SetM$
10: $vecters \leftarrow vecters \cup M[j]$
11: **endfor**

//针对第 j 个符号向量,对所有的被测代码片段进行哈希聚类,生成与脆弱代码相似度大于阈值的特征向量集合对应的被测代码特征矩阵集合 *Matrix*.

12: $Matrix \leftarrow LSH(vecters, M'[j], \delta)$
13: **foreach** M **in** $Matrix$
14: $M.index \leftarrow M.index \cup \{j\}$
15: $group \leftarrow group \cup \{M\}$
16: $SetM_1 \leftarrow SetM_1 - M$
17: **endfor**
18: **endwhile**
19: **endfor**
20: **endfor**
21: **return** *group*

(8) 脆弱性代码检测

被报告为和脆弱代码片段成克隆对关系的被测代码片段有可能是对修补过的脆弱代码的克隆,可按照定义 7 中的脆弱性检测方法进一步提高被测代码片段的脆弱性报告精度.算法较为简单,不再赘述.

3 测试与分析

我们从公开漏洞库中筛选了 131 条脆弱代码,包括 linux kernel[20]中 96 条,dhcp[21]中 11 条,openssl[22]中 8 条,samba[18]中 4 条,net-snmp[23]中 2 条,bind[19]中 10 条,作为 CVdetector 检测工具测试的知识库.表 2 给出了脆弱代码的相关软件信息.

Table 2 CVdetector's vulnerability code knowledge base

表 2 Cvdetector 脆弱代码知识库

克隆知识源	漏洞代码条目
linux kernel	96
dhcp	11
openssl	8
samba	4
net-snmp	2
bind	10

CVdetector 选择与脆弱代码所在的软件功能和开发框架相似的 bftpd[24],dnsmasq[25],openssh[26],ntp[27]和 android-kernel[28]这 5 个代码规模不一的软件对象进行了测试.表 3 给出了各个对象的脆弱点报告数量以及时间开销.

Table 3 Basic information of CVdetector's testing software

表 3 CVdetector 测试软件基本信息

测试软件	版本	脆弱点报告数量(特征矩阵)	时间开销(特征矩阵)	脆弱点报告数量(特征向量)	时间开销(特征向量)
android-kernel	2.3.3	403	29 分 44 秒	2 111	25 分 38 秒
dnsmaq	2.47	123	5 分 14 秒	392	4 分 30 秒
ntp	4.2.2	9	12 分 23 秒	58	9 分 22 秒
openssh	5.8p1	59	8 分 7 秒	67	7 分 6 秒
bftp	1.6	12	1 分 12 秒	12	1 分 10 秒

在表 3 中,

- 脆弱点报告数量(特征矩阵)是指在考虑不同脆弱代码的克隆类型情况下对测试代码片段进行特征矩阵提取,利用代码克隆方法检测出的脆弱点报告数量.
- 脆弱点报告数量(特征向量)是指对测试代码片段进行特征向量提取,即,在不考虑脆弱性类型的情况下,用统一的节点类型构造特征向量,利用代码克隆方法检测出的脆弱点报告数量.

从表 3 的数据可以分析出:特征向量方法的时间开销相对降低了 20%左右,但脆弱点报告数量增加了数倍,极大地增加了后续人工对脆弱点报告进一步审查的工作量.

表 4 给出了 CVdetector 对以上 5 个软件对象的脆弱点具体报告情况,其中,“特征矩阵方法”、“特征向量方法”表示表 2 中的特征矩阵和特征向量两种关键节点抽取方法;而“静态分析方法”代表商业化静态分析工具[29−31]用到的检测方法,“√”表示检测到相应行中的 CVE 漏洞,“×”表示未检测到相应行中的 CVE 漏洞.

Table 4 Clone vulnerability report about CVdetector's testing software

表 4 CVdetector 对被测软件的克隆脆弱性报告

脆弱性	测试软件	克隆公开漏洞	脆弱性类别	所在位置	特征矩阵方法	特征向量方法	静态分析方法
1	Dnsmasq	CVE-2009-2957	堆溢出	****	√	√	√
2	android-kernel	CVE-2009-2692	指针误用	sock_sendpage	√	√	×
3	android-kernel	CVE-2011-4110	指针误用	user_update	√	√	×
4	android-kernel	CVE-2009-2691	条件竞争	mm_formaps	√	√	×
5	android-kernel	CVE-2010-0743	格式串溢出	isns_attr_query	√	√	×
6	android-kernel	CVE-2010-3904	认证缺失	rds_page_copy_user	√	×	×
7	android-kernel	CVE-2010-3442	认证缺失	snd_ctl_new	√	×	×
8	android-kernel	CVE-2010-2538	整数溢出	btrfs_ioctl_clone	√	×	×
9	android-kernel	CVE-2009-1298	不恰当参数	ip_frag_reasm	√	×	×
10	android-kernel	CVE-2009-4895	资源保护不当	tty_fasync	√	√	√

可以看出,在考虑不同脆弱代码的克隆类型情况下,利用特征矩阵方法在 dnsmasq 中发现了 1 个堆溢出相关的脆弱性,在 android-kernel 中发现了 9 个公开漏洞克隆出来的脆弱性,包括条件竞争、认证缺失、整数溢出、不恰当选项等多个类型的脆弱性;在特征矩阵方法检测的上述脆弱点中,利用特征向量方法漏报了其中的 4 个关于认证缺失、不恰当选项、整数溢出的脆弱性,利用商业化静态分析工具检测漏报了其中的 8 个脆弱性.可以看出,本文提出的基于特征矩阵的克隆检测方法在考虑脆弱性代码的语法和语义特征情况下,对脆弱性的检测结果更为理想,降低了漏报率和虚报率.从表 5 可以看出,在 403 个报告中,初步核查即发现 9 处高风险脆弱性,具有较低的虚报率.图 5 给出了 CVdetector 对 5 个目标的测试时间开销.从图 5 可以看出,测试时间和被测软件代码规模基本成线性增长关系.

Table 5 Module distribution of android-kernel vulnerability

表 5 Android-Kernel 脆弱性的模块分布

模块	所在位置	脆弱点数量
文件系统模块	fs	79
内核初始化程序模块	init	0
内存管理程序模块	mm	95
内核管理核心模块	kernel	41
设备驱动模块	drivers	42
网络模块	net	25
核心进程通信模块	ipc	17
体系结构核心模块	arch	39
加密模块	crypto	22
声音系统架构模块	sound	43

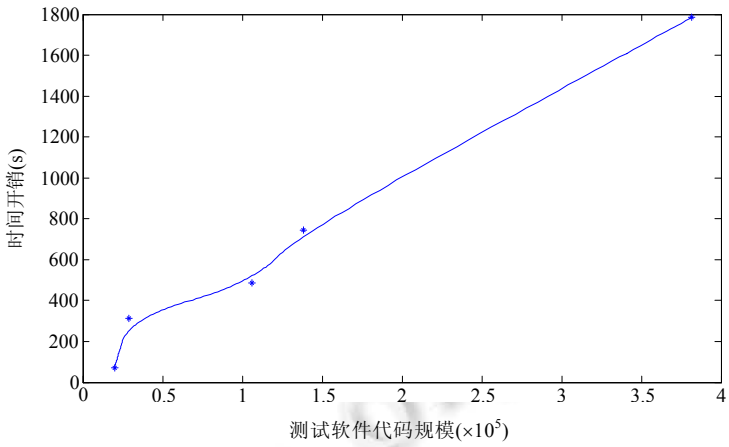

Fig.5 Relationship of CVdetector's testing time and code scale

图 5 CVdetector 测试时间和代码规模的关系

4 结 论

本文通过对源代码脆弱性机制原理分析,提出了一种基于特征矩阵的代码克隆检测方法的多类脆弱性检测模型 Cvdetector.CVdetector 利用对语法分析树的节点类型分析,抽取特定的关键节点类型描述不同的代码克隆类型,通过遍历计算代码片段对应的语法分析树中关键节点的数量,构造代码对应的特征矩阵.模型利用一种半自动化方法从公开漏洞数据库中抽取脆弱性代码作为克隆检测的脆弱代码知识库,通过对同一代码进行多次聚类计算,可以有效降低漏报率.最后,CVdetector 在对多个开源系统软件和应用软件进行的脆弱性检测测试中取得了良好的效果,准确地定位了多个脆弱点.实验结果显示:CVdetector 具有较好的脆弱性类型检测能力,能发现目前主流静态分析工具能力之外的多种脆弱性.

References:

[1] Tripathi A. Towards standardization of vulnerability taxonomy. In: Proc. of 2010 the 2nd Int'l Conf. on Computer Technology and Development. 2010. 379−384. [doi: 10.1109/ICCTD.2010.5645826]

[2] Howard M, LeBlanc D, Viega J, Wrote; Xiao FT, Yang MJ, Trans. 19 Deadly Sins of Software Security Programming Flaws and How to Fix Them. Beijing: Qinghua University Press, 2006 (in Chinese).

[3] Viega J, Bloch JT, Kohno Y, McGraw G. Its4: A static vulnerability scanner for c and c++ code. In: Proc. of the 16th Annual Computer Security Applications Conf. (ACSAC 2000). Washington: IEEE Computer Society, 2000. [doi: 10.1109/ACSAC.2000.898880]

[4] software R. RATs. http://www.securesw.com/rats/

[5] Dwheeler. Flawfinder software. 2007. http://sourceforge.net/projects/flawfinder/

[6] Aiken A, Bugrara S, Dillig I. Saturn project. http://saturn.stanford.edu

[7] Rose/Compass static analysis tools user manual. http://www.rosecompiler.org/compass.pdf

[8] 秦晓军,甘水滔,陈左宁.一种基于一阶逻辑的软件代码安全性缺陷静态检测技术.中国科学:信息科学,2014,44:108−129. [doi: 10.1360/N112013-00095]

[9] Roy CK, Cordy JR, Koschke R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming, 2009,74(7):470−495. [doi: 10.1016/j.scico.2009.02.007]

[10] Nguyen TT, Nguyen HA, Pham NH, Al-Kofahi JM, Nguyen TN. Recurring bug fixes in object oriented programs. In: Proc. of the Int'l Conf. on Software Engineering (ICSE 2010). ACM Press, 2010. 315−324. [doi: 10.1145/1806799.1806847]