软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):2149−2162 [doi: 10.3724/SP.J.1001.2012.04117]

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

基于下推系统可达性分析的程序机密消去机制*

孙 聪[1,2,3+], 唐礼勇[1,2,3], 陈 钟[1,2,3]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](网络与软件安全保障教育部重点实验室(北京大学),北京 100871)

Declassification Enforcement on Program with Reachability Analysis of Pushdown System

SUN Cong[1,2,3+], TANG Li-Yong[1,2,3], CHEN Zhong[1,2,3]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[2](Key Laboratory of Network and Software Security Assurance (Peking University), Ministry of Education, Beijing 100871, China)

+ Corresponding author: E-mail: suncong.pku@gmail.com

Sun C, Tang LY, Chen Z. Declassification enforcement on program with reachability analysis of pushdown system. *Journal of Software*, 2012,23(8):2149−2162 (in Chinese). http://www.jos.org.cn/1000-9825/4117.htm

Abstract: The study proposes a verification mechanism based on reachability analysis of pushdown system to enforce existing declassification policies of language-based information flow security. The pushdown rules of store and match primitives are embedded in the abstract model after compact self-composition. The security property with respect to different declassification policies is violated when the illegal-flow state is reached in the pushdown system. The experimental results show improvement in precision, compared with the type-based mechanisms, and growth in effectiveness compared with the RNI-enforcement based on automated verification.

Key words: information flow security; declassification; pushdown system; automated verification; program analysis

摘 要: 针对程序语言信息流安全领域的现有机密消去策略,提出了一种基于下推系统可达性分析的程序信息流安全验证机制.将存储-匹配操作内嵌于对抽象模型的紧凑自合成结果中,使得对抽象结果中标错状态的可达性分析可以作为不同机密消去策略下程序安全性的验证机制.实例研究说明,该方法比基于类型系统的方法具有更高的精确性,且比已有的自动验证方法更为高效.

关键词: 信息流安全;机密消去;下推系统;自动验证;程序分析

中图法分类号: TP309 **文献标识码**: A

无干扰性[1]作为信息流安全的一般性规约,拒绝信息从高安全级到低安全级的安全级降级(downgrading).而在实际应用中,存在很多需要高安全级信息以预期的方式泄露给低安全级环境的情况,如基本的口令鉴别等,

* 基金项目: 国家自然科学基金(60773163, 60821003, 60872041, 60911140102); 国家科技部重大专项(2011ZX03005-002); 中央高校基本科研业务费专项资金(JY10000903001); 装备预研基金(9140A15040210HK6101)

收稿时间: 2010-10-07; 定稿时间: 2011-09-01

这时,不泄露任何机密性信息的规约将影响系统的可用性.包含安全级降级的程序可能会超出程序员预期地将更多的信息进行安全级降级,亦或攻击者可能违反信息释放策略而提取出额外的机密信息.在这种情况下,如何验证程序是否按照预期规约的方式泄露机密信息就变得十分重要.与保密性信息流安全策略相关的安全级降级称为机密消去(declassification).相应地,与完整性信息流安全策略相关的安全级降级称为签认(endorcement).本文讨论保密性信息流安全策略.

Sabelfeld等人[2]对多种语言和演算上的不同信息释放策略定义进行了分类,策略按照基本目标的不同分为以下4类:(1) 释放什么信息(what);(2) 谁释放信息(who);(3) 信息释放到系统的什么位置(where);(4) 信息何时被释放(when).本文提出的验证机制针对what和where两个目标维度下的安全策略.从what的角度来讲,一个标志性工作是定界释放(delimited release,简称DR)[3].不严格无干扰性(relaxed noninterference,简称RNI)[4]与定界释放紧密相关[5],实际上,不严格无干扰性是在 what 的基础上,要求当特定的语法表达式出现时,在该语法表达式所在的位置依照该语法表达式所给出的形式进行机密消去.此后,在what基础上考虑where的工作还包括局部定界释放(localized delimited release,简称 LDR)[6].从 where 的角度来讲,标志性工作包括非传递无干扰性(intransitive noninterference,简称INI)[7]、不披露性(non-disclosure,简称ND)[8]及渐进释放(gradual release,简称GR)[9].定界不披露性(delimited non-disclosure,简称 DND)[10]和条件渐进释放(conditional gradual release,简称CGR)[11]分别将不披露性和渐进释放扩展到考虑what.Mantel等人[12]给出的WHERE,$WHAT_1$,$WHAT_2$策略不具备可叠加性.现有机密消去策略对应目标及依赖关系如图1所示.

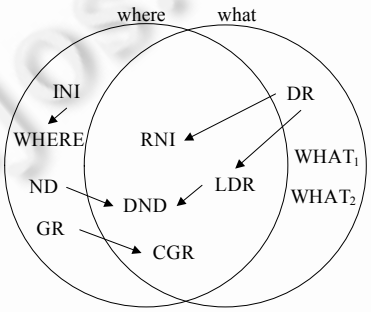

Fig.1 Relationship between policies under the what- and where- declassification goals

图1 what与where目标维度下的机密消去策略及关系

本文并未在what和where目标维度上提出新的安全策略,而是从验证机制的角度讨论如何对现有what和where目标维度上的典型机密消去策略(DR,RNI,LDR,DND,CGR)进行验证.我们提出了一种基于下推系统可达性分析的自动验证方法.该方法建立在我们在此前工作中提出的下推系统紧凑自合成(compact self-composition)[13]及下推系统可达性分析思想[14]基础上,并提出了一种存储-匹配模式以实现被释放表达式及终态低安全级变量之间等价关系的判定.从具体验证方法来看,一方面,现有策略(LDR,DND,CGR)的支持机制主要基于类型系统,而由于基于类型系统的方法保守性较强(Sec.1[15]),本文通过对实例的分析说明了下推系统可达性分析方法相比基于类型系统的方法精确性更高;另一方面,Barthe 等人[15]从自合成理论的角度讨论了定界释放策略的定义方式,而 Terauchi 等人[5]的工作是目前仅有的以自动程序验证作为验证机制(使用BLAST[16]作为工具)来讨论机密消去策略(不严格无干扰性策略)的工作,因而本文还比较了文献[5]方法与本文方法的验证效果.本文假定所讨论的程序执行不存在发散的情况,即将安全策略限定为终止不敏感[5,15],允许由程序终止性引起的信息泄露发生.

本文第1节介绍存储-匹配模式以及在存储-匹配模式基础上如何将下推系统可达性分析用于验证定界释放和不严格无干扰性.第2节介绍方法如何应用于局部定界释放、定界不披露性及条件渐进释放.第3节给出方法评价.第4节给出结论及展望.

1 基于存储-匹配的定界释放及不严格无干扰性

定界释放策略使用安全出口(escape hatch)作为高安全级信息合法释放的途径,在基本格模型 low≺high 上,安全出口表示为 declassify(e),其中,e 为被释放的包含高安全级信息的表达式.高安全级信息不能通过除了安全出口以外的任何其他形式进行泄露.以 L 和 H 分别表示程序状态的低安全级部分和高安全级部分,$(S,e)\Downarrow v$ 表示表达式 e 在状态 S 下的求值为 v,$(S,\mathrm{P})\Downarrow S'$表示程序 P 在初态 S 下的执行终止于终态 S',则定界释放可表述如下:

定义 1(定界释放). 程序 P 在定界释放策略下安全,如果

$$(\forall 0\leqslant i\leqslant n.(LH_1,e_i)\Downarrow v_i\wedge(LH_2,e_i)\Downarrow v_i'\wedge v_i=v_i')\wedge(LH_1,P)\Downarrow(L'H_1')\wedge(LH_2,P)\Downarrow(L''H_2')\Rightarrow L'=L'',$$

其中,$e_i(0\leqslant i\leqslant n)$为程序 P 中包含的所有安全出口 declassify(e_i)所释放的表达式.

与无干扰性定义相比,定界释放定义的命题前件中加入了表达式 e_i 的初态等价.定界释放实际上说明,在初态下,如果高安全级信息的变化没有导致程序将要通过安全出口释放的表达式的值发生变化,那么这种初态高安全级信息的变化就不应该反映为终态低安全级输出的变化;反之,如果终态低安全级输出泄露了高安全级信息,那么这种信息泄露也不会是由表达式 e_i 的初态值不同所引起的.

延用作者此前工作[14]中抽象下推系统上的可达性分析思想来验证程序是否违反定界释放,将抽象模型构造分为 3 部分:(1) 基本自合成;(2) 初态低安全级变量的交叉赋值;(3) 标错状态构造.基本自合成使用此前工作[13]中的紧凑自合成,对下推系统模型进行变量重命名后与原下推系统模型进行合成,并根据下推模型特点忽略对局部变量的重命名,以减小模型状态空间和验证开销.在此更改基本自合成结果,使其包含存储-匹配操作,存储-匹配方式分为两类:一类用于判定终态低安全级变量与其伙伴变量之间的相等关系,分别用 sto/mat 表示;另一类用以确定初态下表达式 e_i 与其伙伴表达式之间的相等关系,分别用_sto/_mat 表示.在基本自合成结果构造出的内存模型基础上,两类存储-匹配操作分别作用于两个全局队列($\boldsymbol{O}$,po)和($\boldsymbol{D}$,pd),其中,$\boldsymbol{O}$,$\boldsymbol{D}$ 为全局数组而全局变量 po,pd 分别指向 $\boldsymbol{O}$,$\boldsymbol{D}$ 中下一个元素存放位置,po,pd 均初始化为 0.对于由基本自合成结果模拟的两次相关的程序执行,第 1 次执行中使用两类存储操作,而第 2 次执行中使用两类匹配操作.存储-匹配操作对应的 Remopla[17]代码和下推规则见表 1.另外,对于布尔类型的表达式值,定义对应的存储-匹配操作_bsto/_bmat,根据布尔型的形参值向($\boldsymbol{D}$,pd)中存入整型值并匹配存入的整型值.

Table 1 Pushdown rules of store-match operations

表 1 存储-匹配下推规则

	Remopla 代码	下推规则	
sto	$O[po]=x$, $po=po+1$;	$q\langle\text{sto}\rangle\to q\langle\text{sto1}\rangle$	$(O'[po]=x\ \&\ po'=po+1\ \&\ (A\ i\ (0,1)\ (D'[i]=D[i]))\ \&$ $pd'=pd\ \&\ (A\ i\ (0,1)\ ((i=po\vert O'[i]=O[i]))\ \&\ x'=x)$
		$q\langle\text{sto1}\rangle\to q\langle\cdot\rangle$	$((A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&\ (A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$
mat	if :: $O[po]!=x$ →goto IFlow; :: else →$po=po+1$; fi	$q\langle\text{mat}\rangle\to q\langle\text{IFlow}\rangle$	$(O[po]!=x\ \&\ (A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&$ $(A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$
		$q\langle\text{mat}\rangle\to q\langle\text{mat1}\rangle$	$(!(O[po]!=x)\ \&\ po'=po+1\ \&\ (A\ i\ (0,1)\ (D'[i]=D[i]))\ \&$ $pd'=pd\ \&\ (A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ x'=x)$
		$q\langle\text{mat1}\rangle\to q\langle\cdot\rangle$	$((A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&\ (A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$
_sto	$D[pd]=x$, $pd=pd+1$;	$q\langle\text{\_sto}\rangle\to q\langle\text{\_sto1}\rangle$	$(D'[pd]=x\ \&\ pd'=pd+1\ \&\ (A\ i\ (0,1)\ ((i=pd\vert D'[i]=D[i])))\ \&$ $(A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po\ \&\ x'=x)$
		$q\langle\text{\_sto1}\rangle\to q\langle\cdot\rangle$	$((A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&\ (A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$
_mat	if :: $D[pd]!=x$ →goto NOPRE; :: else →$pd=pd+1$; fi	$q\langle\text{\_mat}\rangle\to q\langle\text{NOPRE}\rangle$	$(D[pd]!=x\ \&\ (A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&$ $(A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$
		$q\langle\text{\_mat}\rangle\to q\langle\text{\_mat1}\rangle$	$(!(D[pd]!=x)\ \&\ pd'=pd+1\ \&\ (A\ i\ (0,1)\ (D'[i]=D[i]))\ \&$ $(A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po\ \&\ x'=x)$
		$q\langle\text{\_mat1}\rangle\to q\langle\cdot\rangle$	$((A\ i\ (0,1)\ (D'[i]=D[i]))\ \&\ pd'=pd\ \&\ (A\ i\ (0,1)\ (O'[i]=O[i]))\ \&\ po'=po)$

以文献[3]中的 Avg-Attack 为例(为了简化,取两个高安全级变量),其抽象结果见表 2 左上部,自合成结果见表 2 右侧.区别于紧凑自合成,这里并不把形如过程调用的 declassify 抽象为形如 $q\langle n_i\rangle\to q\langle\text{declassify0 }n_j\rangle$,而是首先获取其实参形式($h1$+$h2$)/2,直接放入全局变量 $iret$ 中临时保存,如表 2 中箭头所示.另一与紧凑自合成不同的是对 main 函数返回规则 $q\langle\text{DEFAULT_3_}\rangle\to q\langle\cdot\rangle$的处理,在自合成时改为分别调用 sto 和 mat 进行低安全级终态等价性比较.另外,在两次执行开始处还要重置数组索引,在首次执行开始时进行初态下低安全级变量与其伙伴

变量之间的交叉赋值操作.由于在预处理 declassify 时已经获得了所有实参的形式,因而可以向两次执行开始处加入表示定界释放命题前件的存储-匹配下推规则来表达 e_i 的初态等价.

Table 2 Abstract model of Avg-Attack and the result of self-composition

表 2 Avg-Attack 的抽象模型及自合成结果

$q\langle$main$\rangle \rightarrow q\langle$DEFAULT_1_$\rangle$ ($h1'=h2$ & $h2'=h2$ & $avg'=avg$)	$q\langle$main$\rangle \rightarrow q\langle$DEFAULT_1_$\rangle$	($h1'=h2$ & $h2'=h2$ & $avg'=avg$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1t′=h1t*** & ***h2t′=h2t*** & ***avgt′=avgt***)
	$q\langle$maint$\rangle \rightarrow q\langle$DEFAULT_1_$t\rangle$	($h1t'=h2t$ & $h2t'=h2t$ & $avgt'=avgt$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1′=h1*** & ***h2′=h2*** & ***avg′=avg***)
$q\langle$DEFAULT_1_$\rangle \rightarrow$ $q\langle$declassify DEFAULT_2_$\rangle$ ($x'=(h1+h2)/2$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$)	$q\langle$DEFAULT_1_$\rangle \rightarrow q\langle$DEFAULT_2_$\rangle$	($iret'=(h1+h2)/2$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1t′=h1t*** & ***h2t′=h2t*** & ***avgt′=avgt***)
q*⟨DEFAULT_1_⟩→*q*⟨DEFAULT_2_⟩** (iret′*=(*h1*+*h2*)/2 & *h1′=h1* & *h2′=h2* & *avg′=avg***)	$q\langle$DEFAULT_1_$t\rangle \rightarrow q\langle$DEFAULT_2_$t\rangle$	($iret'=(h1t+h2t)/2$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1′=h1*** & ***h2′=h2*** & ***avg′=avg***)
$q\langle$DEFAULT_2_$\rangle \rightarrow q\langle$DEFAULT_3_$\rangle$ ($avg'=iret$ & $h1'=h1$ & $h2'=h2$)	$q\langle$DEFAULT_2_$\rangle \rightarrow q\langle$DEFAULT_3_$\rangle$	($avg'=iret$ & $h1'=h1$ & $h2'=h2$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1t′=h1t*** & ***h2t′=h2t*** & ***avgt′=avgt***)
	$q\langle$DEFAULT_2_$t\rangle \rightarrow q\langle$DEFAULT_3_t$\rangle$	($avgt'=iret$ & $h1t'=h1t$ & $h2t'=h2t$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1′=h1*** & ***h2′=h2*** & ***avg′=avg***)
$q\langle$DEFAULT_3_$\rangle \rightarrow q\langle\cdot\rangle$ ($h1'=h1$ & $h2'=h2$ & $avg'=avg$)	$q\langle$DEFAULT_3_$\rangle \rightarrow q\langle$**sto END_**$\rangle$	($x'=avg$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1t′=h1t*** & ***h2t′=h2t*** & ***avgt′=avgt***)
	$q\langle$DEFAULT_3_$t\rangle \rightarrow q\langle$**mat END_*t***$\rangle$	($x'=avgt$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$ & (***A i*** (**0**,**1**) (***D′*** [***i***]=***D***[***i***])) & ***pd′=pd*** & (***A i*** (**0**,**1**) (***O′*** [***i***]=***O***[***i***])) & ***po′=po*** & ***h1′=h1*** & ***h2′=h2*** & ***avg′=avg***)
交叉赋值,数组系数初始化	$q\langle$**tmp_main**$\rangle \rightarrow q\langle$DEFAULT_5_$\rangle$	(***avgt′=avg*** & (A i (0,1) ($D'[i]=D[i]$)) & ***pd′*=0** & (A i (0,1) ($O'[i]=O[i]$)) & ***po′*=0** & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$)
	$q\langle$**END_**$\rangle \rightarrow q\langle$DEFAULT_5_*t*$\rangle$	((A i (0,1) ($D'[i]=D[i]$)) & ***pd′*=0** & (A i (0,1) ($O'[i]=O[i]$)) & ***po′*=0** & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)
定界释放命题前件存储-匹配	$q\langle$DEFAULT_5_$\rangle \rightarrow q\langle$**_sto main**$\rangle$	($x'=(h1+h2)/2$ & (A i (0,1) ($D'[i]=D[i]$)) & $pd'=pd$ & (A i (0,1) ($O'[i]=O[i]$)) & $po'=po$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)
	$q\langle$DEFAULT_5_*t*$\rangle \rightarrow q\langle$**_mat maint**$\rangle$	($x'=(h1t+h2t)/2$ & (A i (0,1) ($D'[i]=D[i]$)) & $pd'=pd$ & (A i (0,1) ($O'[i]=O[i]$)) & $po'=po$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)
特殊状态	$q\langle$**END_*t***$\rangle \rightarrow q\langle$**END_*t***$\rangle$	((A i (0,1) ($D'[i]=D[i]$)) & $pd'=pd$ & (A i (0,1) ($O'[i]=O[i]$)) & $po'=po$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)
	$q\langle$**NOPRE**$\rangle \rightarrow q\langle$**NOPRE**$\rangle$	((A i (0,1) ($D'[i]=D[i]$)) & $pd'=pd$ & (A i (0,1) ($O'[i]=O[i]$)) & $po'=po$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)
	$q\langle$**IFlow**$\rangle \rightarrow q\langle$**IFlow**$\rangle$	((A i (0,1) ($D'[i]=D[i]$)) & $pd'=pd$ & (A i (0,1) ($O'[i]=O[i]$)) & $po'=po$ & $h1'=h1$ & $h2'=h2$ & $avg'=avg$ & $h1t'=h1t$ & $h2t'=h2t$ & $avgt'=avgt$)

NOPRE 状态表示当两个相关的初态下安全出口要释放的表达式 e_i 不相等时所进入的状态,进入 NOPRE 则意味着标错状态 IFlow 不可达,因而也就使得 IFlow 的可达性蕴含着"两初态下 e_i 相等"这一定界释放的命题前件.为解释方便起见,将此后生成的下推系统都恢复为 Remopla 表示,表 2 自合成结果可恢复如下:

- *avgt*=*avg*,*pd*=0,*po*=0; _sto((*h*1+*h*2)/2); *h*1=*h*2; *avg*=(*h*1+*h*2)/2; sto(*avg*).
- *pd*=0,*po*=0; _mat((*h*1*t*+*h*2*t*)/2); *h*1*t*=*h*2*t*; *avgt*=(*h*1*t*+*h*2*t*)/2; mat(*avgt*).

由第 2 类存储-匹配所保证的初态高安全级变量平均值的不可分辨性在 *h*1=*h*2 后无法保证低安全级变量 *avg* 与其伙伴变量 *avgt* 之间的不可分辨性,从而由第 1 类匹配 mat 到达标错状态 IFlow,因而 Avg-Attack 在定界释放策略下不安全.使用 Moped[18]对抽象模型进行可达性检测,由 Moped 输出的 IFlow 的一个可达路径见表 3.

Table 3 Reachability trace to the illegal-flow state (IFlow) of Avg-Attack

表 3 Avg-Attack 的标错状态(IFlow)可达路径

构型	抽象全局变量	抽象局部变量
q⟨tmp_main⟩	*D*=[0,0]; *pd*=0; *O*=[0,0]; *po*=0; *h*1=2; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_5_⟩	*D*=[0,0]; *pd*=0; *O*=[0,0]; *po*=0; *h*1=2; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨_sto main⟩	*D*=[0,0]; *pd*=0; *O*=[0,0]; *po*=0; *h*1=2; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨_sto1 main⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=2; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨main⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=2; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_1_⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=1; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_2_⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=1; *h*2=1; *avg*=0; *h*1*t*=0; *h*2*t*=3; *avgt*=0; *iret*=1	
q⟨DEFAULT_3_⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨sto END_⟩	*D*=[1,0]; *pd*=1; *O*=[0,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨sto1 END_⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=1; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨END_⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=1; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_5_*t*⟩	*D*=[1,0]; *pd*=0; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨_mat maint⟩	*D*=[1,0]; *pd*=0; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨_mat1 maint⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	*x*=1
q⟨maint⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=0; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_1_*t*⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=3; *h*2*t*=3; *avgt*=0	
q⟨DEFAULT_2_*t*⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=3; *h*2*t*=3; *avgt*=0; *iret*=3	
q⟨DEFAULT_3_*t*⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=3; *h*2*t*=3; avgt=3	
q⟨mat END_t⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=3; *h*2*t*=3; *avgt*=3	*x*=3
q⟨IFlow END_*t*⟩	*D*=[1,0]; *pd*=1; *O*=[1,0]; *po*=0; *h*1=1; *h*2=1; *avg*=1; *h*1*t*=3; *h*2*t*=3; *avgt*=3	

Terauchi 等人[5]从理论上说明了自合成可处理不严格无干扰性,并将不严格无干扰性从函数式语言扩展到命令式语言.不严格无干扰性实际上将定界释放中需要通过安全出口释放的表达式的求值函数作为安全策略应用于实际的状态变量值,与定界释放类似,这一对求值函数的应用仍然是在初态完成.比定界释放更灵活的是,求值函数的形式可以是复杂的程序语句.

定义 2(不严格无干扰性). 假定程序 P 中包含的所有安全策略为 $f_1,f_2,\ldots,f_m$,其中,f_i 应用于状态 S 的结果为 $f_i\ S(x_{i,1})\ S(x_{i,2})...S(x_{i,t_i})$,$x_{i,j}$ 为程序变量,t_i 为 f_i 中受限变量的个数.程序 P 在不严格无干扰性策略下安全,如果

$$(LH_1,\mathrm{P})\Downarrow L'H_1'\wedge(LH_2,\mathrm{P})\Downarrow(L''H_2')\wedge(\forall 1\leqslant i\leqslant m.(f_i\ LH_1(x_{i,1})\ LH_1(x_{i,2})\ \ldots\ LH_1(x_{i,t_i}))=$$
$$(f_i\ LH_2(x_{i,1})\ LH_2(x_{i,2})\ldots\ LH_2(x_{i,t_i})))\Rightarrow L'=L''.$$

定义实际上指出了不严格无干扰性要求策略应用于低安全级等价的不同初态上的效果总相同.当 f_i 为λ演算形式时,不要求 f_i 为封闭的,即 f_i 中仍可存在作为自由变元的变量.考虑文献[5]中 Fig.3 中的例子(见图 2 中的 P1),使用本文方法需将策略 *f*=λ*x*.if (hashfunc(*input*)=*hash*) then *x* else *c* 显式实现为:

```
module int f(int x){int tmp;
    tmp=hashfunc(input);
    if:: tmp==hash→return x; :: else→return c; fi
}
```

这时,表 2 所示抽象过程中命题前件存储匹配变为如下代码对应的下推规则:

... tmp=*f*(*h*); _sto(tmp); ... tmp=*ff*(*ht*); _mat(tmp); ...

f 需要显式地指派为策略函数以区分于一般函数.此时,ff 对应的是 f 的抽象结果进行紧凑自合成得到的一系列下推规则.策略中允许自由变元,实际上要求 *input* 和 *hash* 均为全局变量.

P1: **if** (hashfunc(*input*) = *hash*) **then** $t := t+1$; $l := l + secret$; **else** skip;
P2: $l :=$ declassify(h); c; $l := h$;
P3: $l := h$; c; $l :=$ declassify(h);
P4: $h1 := h2$; $l :=$ declassify($h1$);
P5: **if** h **then** $l :=$ declassify($h1$) **else** skip;
P6: $h1 := h2$; $h2 := 0$; $l :=$ declassify($h2$); $h2 := h1$; $l := h2$;
P7: $h2 := 0$; **if** $h1$ **then** $l :=$ declassify($h1$); **else** $l :=$ declassify($h2$);
P8: $l := 0$; (**if** l **then** $l :=$ declassify(h) **else** skip); $l := h$;
P9: $h2 := 0$; **if** $h1$ **then** $l :=$ declassify($h2$); **else** $l := 0$;
P10: $l :=$ declassify($h! = 0$); **if** l **then** $l1 :=$ declassify($h1$) **else** skip;
P11: **if** true **then** $l := h$ **else** $l :=$ declassify(h);

Fig.2 Program cases
图 2 程序用例

2 基于存储-匹配的局部定界释放、定界不披露性及条件渐进释放

2.1 局部定界释放

局部定界释放的提出基于以下考虑:机密信息在程序执行过程中的什么位置被释放,将影响到程序其余部分对机密信息的使用是否违反信息流安全性.在程序执行过程中,如果机密信息在某一状态已被显式地释放,则该机密信息在该状态的后续状态下可以直接赋给低安全级变量,而不需要显式地使用机密消去原语;而对于在某一状态尚未被释放的机密信息,在该机密信息被机密消去原语显式地释放之前,都不能直接将其赋给低安全级变量.可见,对于局部定界释放,程序执行的中间状态非常重要.

对于程序执行的任一中间状态,将其看作程序前缀执行的终态.如果将定义 1 中的定界释放看作初态和终态之间的关系,那么对此中间状态,作为前缀执行的终态,要求前缀执行的初态与其满足定界释放关系.而对于前缀执行的初态选择,仍选择程序的初态.这样,实际上用 declassify 语句对定界释放所定义的命题前件进行了区分,位于不同 declassify 语句对之间的中间状态(前缀执行的终态)对应的命题前件不同.

假定程序状态形如$\langle LH,E\rangle$,其中,集合 E 保存程序此前的执行过程中已释放的表达式.考虑程序片段…c; declassify(e);c';…的执行:

$$(\langle L_0H_0,\varnothing\rangle,...c;\text{declassify}(e);c';) \to^* (\langle L_1H_1,E_1\rangle,c;\text{declassify}(e);c';) \to$$
$$(\langle L_1'H_1',E_1\rangle,\text{declassify}(e);c';) \to (\langle L_2H_2,E_1\cup\{e\}\rangle,c';) \to^* ...$$

对应于 c 和 c'的定界释放命题前件中的表达式集合分别为 E_1 和 $E_1\cup\{e\}$.在前缀执行的初态和终态均确定的基础上,即可进行两者的定界释放关系判定.局部定界释放实际上要求对这些对应于不同中间状态的局部的定界释放关系均判定为成立,具体定义如下:

定义 3(局部定界释放). 假定程序 P 的两次相关执行分别形如:

$$(\langle L_0H_0,\varnothing\rangle,\mathrm{P}) \to^* (\langle L_1H_1,\varnothing\rangle,c_1;\mathrm{P}_1) \to^* (\langle L_2H_2,E_1\rangle,c_2;\mathrm{P}_2) \to^* ... \to^* (\langle L_kH_k,\bigcup_{i=1}^{k-1}E_i\rangle,c_k;\mathrm{P}_k) \to^* (\langle L_fH_f,\bigcup_{i=1}^{k}E_i\rangle,\varepsilon).$$

$$(\langle L_0'H_0',\varnothing\rangle,\mathrm{P}) \to^* (\langle L_1'H_1',\varnothing\rangle,c_1';\mathrm{P}_1') \to^* (\langle L_2'H_2',E_1'\rangle,c_2';\mathrm{P}_2') \to^* ... \to^* (\langle L_t'H_t',\bigcup_{i=1}^{t-1}E_i'\rangle,c_t';\mathrm{P}_t') \to^* (\langle L_f'H_f',\bigcup_{i=1}^{t}E_i'\rangle,\varepsilon).$$

其中,$c_1,c_2,...,c_k,c_1',c_2',...,c_t'$ 为执行路径上所有包含 declassify 的语句,$E_0 = E_0' = \varnothing,E_1,E_2,...,E_k,E_1',E_2',...,E_t'$ 分别为 $c_1,c_2,...,c_k,c_1',c_2',...,c_t'$ 释放的机密信息表达式的集合.又假定 $L_{j,j+1}H_{j,j+1}$ 为 L_jH_j 与 $L_{j+1}H_{j+1}$ 之间的任一中间状态,$L_{j,j+1}'H_{j,j+1}'$ 为另一执行上与 $L_{j,j+1}H_{j,j+1}$ 对应的中间状态.程序 P 在局部定界释放策略下安全,如果以下关系成立:

$$(t=k)\wedge\left(\forall 0\leqslant j\leqslant k.\left(L_0=L_0'\wedge\bigcup_{i=0}^{j}E_i=\bigcup_{i=0}^{j}E_i'\wedge_{e_s\in\bigcup_{i=0}^{j}E_i}L_0H_0(e_s)=L_0'H_0'(e_s)\Rightarrow L_{j,j+1}=L_{j,j+1}'\right)\right).$$

以上局部定界释放定义相比原始定义[6]更严格,是一种与程序执行路径相关的定义,而基于互模拟的原始定义与具体程序路径无关.考虑图 2 中的程序 P3,根据局部定界释放定义,$l:=h$ 即违反定义 3 中 $j=0$ 时的关系.而对于程序 P7,当两次相关执行均取 else 分支时,程序终态对应的定界释放命题前件中,机密信息表达式为{$h2$};而在判断定界释放时,由于只与初态和终态相关,程序执行可能采取不同的路径,从而使得非零 $h1$ 值被泄露.程序 P8 的终态对应的定界释放命题前件中可能不存在任何机密信息表达式,这时,局部定界释放判定类似于无干扰性,从而得出不安全.

下面给出使用下推系统可达性分析的局部定界释放验证机制.

在定义 3 中,$L_{j,j+1}H_{j,j+1}$ 的选取在两个 declassify 语句之间具有任意性.而在机制实现时,使用一种粗粒度的方法:找到所有分支结构的汇聚点,当两个 declassify 之间(或最后一个 declassify 与终态 L_fH_f 之间)不存在分支结构的汇聚点时,将 $L_{j,j+1}H_{j,j+1}$ 取为后一个 declassify 之前的那个状态,即 $L_{j+1}H_{j+1}$(或终态 L_fH_f);而当两个 declassify 之间(或最后一个 declassify 与终态 L_fH_f 之间)存在分支结构的汇聚点时,将 $L_{j,j+1}H_{j,j+1}$ 取为当前分支所在的汇聚点之前的状态.而当 $L_{j,j+1}H_{j,j+1}$ 在首个 declassify 之前时,则取为 L_1H_1.由于局部定界释放需要进行多次定界释放判断,因而需要构造多个抽象模型,对每个抽象模型分别判断其标错状态可达性.当每个抽象模型均验证为安全时,原程序才满足局部定界释放.以程序 P2,P3,P7,P8 为例,抽象及验证结果见表 4.在单个抽象模型对应的 Remopla 代码中,省略了初始状态下低安全级变量的交叉赋值以及数组索引归零.$Reach(M_i)$表示对抽象模型 M_i 中标错状态 IFlow 进行可达性检测的结果,√/×分别表示不可达(安全)/可达(不安全).$RA=\wedge Reach(M_i)$为可达性分析的整体结果.P3 作为在定界释放策略下判定为安全的程序,在此可验证出对于局部定界释放是不安全的,因为在抽象模型 P3_1 中,初态 $h\neq ht$ 导致在 mat(lt)中 IFlow 可达.对于抽象模型 P7_3,初态下 $h2=h2t$ 不能保证两个 if 语句进入同一分支,而当初态 $h1\neq 0$ 且 $h1t=0$ 时,mat 将 lt 值与非零不确定值相比较并使得 IFlow 状态可达.

Table 4 Abstract models in Remopla for LDR and the corresponding verification results

表 4 针对局部定界释放的抽象结果(表示为 Remopla)及对应验证结果

用例	抽象模型 M_i	M_i 对应的 Remopla 代码	$Reach(M_i)$	RA
P2	P2_1	sto(l); l=h; c; l=h; … mat(lt); lt=ht; c; lt=ht;	√	√
	P2_2	_sto(h); l=h; c; l=h; sto(l); ... _mat(ht); lt=ht; c; lt=ht; mat(lt);	√	
P3	P3_1	l=h; c; sto(l); l=h; ... lt=ht; c'; mat(lt); lt=ht;	×	×
	P3_2	_sto(h); l=h; c; l=h; sto(l); ... _mat(ht); lt=ht; c'; lt=ht; mat(lt);	√	
P7	P7_1	$h2$=0; if :: $h1$!=0→sto(l); l=$h1$; :: else →sto(l); l=$h2$; fi ... $h2t$=0; if :: $h1t$!=0→mat(lt); lt=$h1t$; :: else →mat(lt); lt=$h2t$; fi	√	×
	P7_2	_sto($h1$); $h2$=0; if :: $h1$!=0→l=$h1$; sto(l); :: else →l=$h2$; fi ... _mat($h1t$); $h2t$=0; if :: $h1t$!=0→lt=$h1t$; mat(lt); :: else →lt=$h2t$; fi	√	
	P7_3	_sto($h2$); $h2$=0; if :: $h1$!=0→l=$h1$; :: else →l=$h2$; sto(l); fi ... _mat($h2t$); $h2t$=0; if :: $h1t$!=0→lt=$h1t$; :: else →lt=$h2t$; mat(lt); fi	×	
P8	P8_1	l=0; if :: l!=0→sto(l); l=h; :: else →sto(l); skip; fi l=h; ... lt=0; if :: lt!=0→mat(lt); lt=ht; :: else →mat(lt); skip; fi lt=ht;	√	×
	P8_2	_sto(h); l=0; if :: l!=0→l=h; sto(l); :: else →skip; fi l=h; ... _mat(ht); lt=0; if :: lt!=0→lt=ht; mat(lt); :: else →skip; fi lt=ht;	√	
	P8_3	l=0; if :: l!=0→l=h; :: else →skip; sto(l); fi l=h; ... lt=0; if :: lt!=0→lt=ht; :: else →skip; mat(lt); fi lt=ht;	√	
	P8_4	_sto(h); l=0; if :: l!=0→l=h; :: else →skip; fi l=h; sto(l); ... _mat(ht); lt=0; if :: lt!=0→lt=ht; :: else →skip; fi lt=ht; mat(lt);	√	
	P8_5	l=0; if :: l!=0→l=h; :: else →skip; fi l=h; sto(l); ... lt=0; if :: lt!=0→lt=ht; :: else →skip; fi lt=ht; mat(lt);	×	

2.2 定界不披露性

定界不披露性[10]与局部定界释放的相似之处在于在全局策略的基础上加入了局部策略.对于定界不披露性,局部策略的作用是对程序执行过程中的中间状态判定条件进行重置. $s'_{i'}\sim_{\Gamma[entry]}t'_{j'}$ 实际上是当前状态对应于初态低安全级变量的低安全级等价.定界不披露性与局部定界释放的不同之处包括以下几个方面:

1. 在局部定界释放定义中,E 中表达式的等价性是在初态下计算的,而定界不披露性则是在实际释放发生前的状态下计算的.

2. 局部定界释放中,E 随着程序的执行具有积累性.而定界不披露性定义中,针对每一对当前状态 s_i,t_j 的 $\Gamma[i]$是局部的,不具有积累性.即,在当前局部策略下,变量 h 的安全级由 declassify(h) in {c}降为低安全级,而在 c 执行过后的状态下 h 又恢复为高安全级变量.例如,文献[10]中 Example 1 的$\Gamma[3](h)$与$\Gamma[2](h)$相比又恢复了机密性.
3. 局部定界释放支持表达式的直接释放.定界不披露性的局部策略不支持任意表达式的直接释放,而仅局限于高安全级变量的释放.

在使用可达性分析验证定界不披露性时,由于局部策略的形式的原因,不能如表 2 所示将 declassify 作为特殊的过程调用处理而在抽象模型中隐式地自动生成存储-匹配操作,此处在 Remopla 程序中显式地加入_sto 和 sto 操作,而在抽象模型中通过自合成过程隐式地自动生成_mat 和 mat 操作.

假设在局部策略 declassify(h) in {c}的 c 中存在语句 l:=h,则在该语句之前使用_sto/_mat 进行高安全级变量的等价性计算,而在单步执行结束时依照初态下变量安全级进行对应于 $s'_{i'} \sim_{\Gamma[entry]} t'_{j'}$ 的低安全级等价性判断.由于在局部策略结束时,被该局部策略释放的变量又恢复高安全级,因而在局部策略结束时需要重新对 h 赋以不确定值,从而结束此前局部策略起始处_sto(h)/_mat(ht)的影响.若局部策略延伸到程序末尾,则后续 h 值不会再影响到低安全级变量,这时不需要重新对 h 赋以不确定值.

下面举例说明模型抽象方法.图 2 中 P3 和 P7 分别对应于文献[10]中 Example 3 和 Example 4.抽象结果用 Remopla 表示,见表 5,RA 仍为可达性分析结果.类似于表 4,在此省略初态低安全级变量交叉赋值及数组索引归零.考虑程序 P2 的抽象结果,$nd1$ 和 $nd2$ 为互不相关的不确定性变量,由于_sto/_mat 的原因,第 1 个 mat(lt)中标错状态不可达,而在第 2 个 mat(lt)中,由于此前 h 和 ht 再次变为互不相关,因而将到达标错状态.文献[10]的 Example 6 与 P2 类似.对于程序 P3,首句的显式信息流将导致抽象模型的验证过程在第 1 个 mat(lt)中到达标错状态.再如 P7,当初始 $h1\neq 0$ 且 $h1t=0$ 时,通过_sto($h1$)有 $D[0]\neq 0$,因而在_mat($h2t$)中总到达 NOPRE,从而 IFlow 不可达;而当 $h1=0$ 且 $h1t\neq 0$ 时,类似地,_mat($h1t$)亦总能到达 NOPRE.由于 P3 和 P7 中局部策略延伸至程序末尾,故无须在局部策略终止处再对高安全级变量赋以不确定值.

Table 5 Abstract models in Remopla for DND and the corresponding verification results

表 5 针对定界不披露性的抽象结果(表示为 Remopla)及对应验证结果

用例	抽象模型	RA
P2	_sto(h); l=h; sto(l); h=$nd1$; l=h; sto(l); ... _mat(ht); lt=ht; mat(lt); ht=$nd2$; lt=ht; mat(lt);	×
P3	l=h; sto(l); _sto(h); l=h; sto(l); ... lt=ht; mat(lt); _mat(ht); lt=ht; mat(lt);	×
P7	$h2$=0; sto(l); if:: $h1$→_sto($h1$); l=$h1$; sto(l); :: else →_sto($h2$); l=$h2$; sto(l); fi ... $h2t$=0; mat(lt); if:: $h1t$→_mat($h1t$); lt=$h1t$; mat(lt); :: else →_mat($h2t$); lt=$h2t$; mat(lt); fi	√

2.3 条件渐进释放

渐进释放[9]作为以 where 为目标的信息流安全策略,其规约基于对攻击者知识的比较.如果当前状态由初态 L_0H_0 执行得到,那么当前状态对应的攻击者知识对应于初态下 H_0 的不确定性,攻击者知识越多,这种不确定性越小.而初态 L_0H_0 本身所对应的攻击者知识为使得由初态起始的执行可终止的所有 H_0 组成的集合.随着程序的执行和高安全级信息的释放,当前状态对应的初态 H_0 的不确定性减小,因而攻击者知识单调递增.

Askarov 等人[9]定义的低安全级事件(low event)形如 l:=e 或 l:=declassify(e).假定程序 P 由初态 L_0H_0 起始的执行路径形如 $(L_0H_0,\mathrm{P}) \to^* (L_1H_1,\ell_1;\mathrm{P}_1) \to^* \ldots \to^* (L_{k-1}H_{k-1},\ell_{k-1};\mathrm{P}_{k-1}) \to^* (L_kH_k,\ell_k;\mathrm{P}_k) \to (L'H',\mathrm{P}_k)$,其中, $\vec{\ell}_k$ 为路径上所有低安全级事件组成的序列.ℓ_k 执行前和执行后的攻击者知识分别为

$$K(L_0,\vec{\ell}_{k-1}) = \{H \mid (L_0H,\mathrm{P}) \xrightarrow{\vec{\ell}_{k-1}}{}^* (L_kH_k,\ell_k;\mathrm{P}_k)\}, K(L_0,\vec{\ell}_k) = \{H \mid (L_0H,\mathrm{P}) \xrightarrow{\vec{\ell}_k}{}^* (L'H',\mathrm{P}_k)\}.$$

当ℓ_k形如 l:=e 时,由渐进释放定义(文献[9]中的 Def.2)及攻击者知识的单调性可知,渐进释放要求:

$$K(L_0,\vec{\ell}_{k-1}) \subseteq K(L_0,\vec{\ell}_k).$$

而当ℓ_k形如 l:=declassify(e)时,渐进释放定义并未给出如何表示实际泄露的信息的多少.用

$$R_0=\{H'\mid\exists H\in K(L_0,\vec{\ell}_{k-1}).(L_0H,\mathrm{P})\xrightarrow{\vec{\ell}_{k-1}}{}^{*}(L_kH_k,\ell_k;\mathrm{P}_k)\wedge(L_0H')\xrightarrow{\vec{\ell}_{k-1}}{}^{*}(L'_kH'_k,\ell_k;\mathrm{P}_k)\wedge L_kH_k(e)=L'_kH'_k(e)\}$$

来表示由 $L_kH_k(e)=L'_kH'_k(e)$ 规约的实际被ℓ_k 释放的信息,规约保证了释放的安全性.在条件渐进释放的定义(文献[11]中的 Def.5.5)中,R 实际上是对 R_0 的扩展,要求在前置条件ρ下才可以进行释放,即

$$R=\{H'\mid\exists H\in K(L_0,\vec{\ell}_{k-1}).(L_0H,\mathrm{P})\xrightarrow{\vec{\ell}_{k-1}}{}^{*}(L_kH_k,\ell_k;\mathrm{P}_k)\wedge(L_0H')\xrightarrow{\vec{\ell}_{k-1}}{}^{*}$$
$$(L'_kH'_k,\ell_k;\mathrm{P}_k)\wedge L_kH_k(e)=L'_kH'_k(e)\wedge L_kH_k\models\rho\wedge L'_kH'_k\models\rho\}.$$

从验证机制上讲,使用第 2 类存储匹配_sto/_mat 来保证在执行 l:=declassify(e)之前的状态,能够使得对应初态满足 R_0;而在 l:=e 之前对表达式 e 使用第 1 类存储-匹配 sto/mat 保证赋值过程,没有导致攻击者知识对应的初态不确定性降低.为了支持 R 中的前置条件ρ,_sto/_mat 需要分别作如下改动:

_sto 更改为

if :: ! $\rho\rightarrow$goto NOPRE; :: else $\rightarrow$skip; fi

$D[pd]=x$, $pd=pd+1$;

而_mat 更改为:

if :: ! $\rho\rightarrow$goto NOPRE; :: else $\rightarrow$skip; fi

if :: $D[pd]!=x\rightarrow$goto NOPRE; :: else $\rightarrow pd=pd+1$; fi

这里,ρ是一个抽象的断言,实际实现时需要根据具体的断言选择实现方法.而当两次相关执行均到达终态时,类似于第 1 节,还要使用第 1 类存储-匹配进行终态之间的低安全级等价性比较.考虑文献[11]中 Sec.5 的例(4):

$l0$:=declassify($h\geqslant 0$);$l1$:=declassify($h\leqslant 0$);抽象结果用 Remopla 表示为

- _bsto($h\geqslant 0$); $l0$=($h\geqslant 0$); _bsto($h\leqslant 0$); $l1$=($h\leqslant 0$); bsto($l0$); bsto($l1$); ...
- _bmat($ht\geqslant 0$); $l0t$=($ht\geqslant 0$); _bmat($ht\leqslant 0$); $l1t$=($ht\leqslant 0$); bmat($l0t$); bmat($l1t$);

在此抽象下标错状态不可达.再如程序 P6,抽象结果用 Remopla 表示为

- $h1=h2$; $h2=0$; _sto($h2$); $l=h2$; $h2=h1$; sto($h2$); $l=h2$; sto(l); ...
- $h1t=h2t$; $h2t=0$; _mat($h2t$); $lt=h2t$; $h2t=h1t$; mat($h2t$); $lt=h2t$; mat(lt);

_sto/_mat 并未降低初态 $h2$ 的不确定性,因而在 mat($h2t$)调用中到达标错状态.

3 实现及评价

本文的工具在此前工具[13]的基础上实现,基本自合成方法(见表 2 的上左至上右的过程)使用紧凑自合成[13].模型抽象的实现内嵌于 Moped v2 的输入语言 Remopla 的语法分析器中(约 7kLOC Lex/Yacc/C 代码).实验环境为 1.66GHz×2 Intel CPU/1GB RAM/Linux 2.6.27-15-generic.实验目的包括以下两个方面:

(1) 通过对相关工作中的实例进行验证,比较相关工作中给出的基于类型系统的方法与本文给出的基于下推系统可达性分析的方法的相对精确性.

(2) 作为使用自动验证的方法,比较本文方法(使用 Moped)与文献[5]方法(使用 BLAST)对不严格无干扰性的验证效果.

图 2 中程序 P1~P11 均为相关工作中的用例程序,出处见表 6.在表 6 中,Sf 表示按照相应的安全策略定义得出的程序安全性(√表示安全,×表示不安全),Ty 表示使用相关工作中给出的类型系统对程序是否为良类型作出的判断(√表示良类型,×表示非良类型).对于某个安全策略,若 Sf 与 Ty 一致,则表示类型系统判定精确;若 Sf/Ty 为√/×,则反映出基于类型系统的方法的保守性;若 Sf/Ty 为×/√,则表示存在漏判.

在表 6 中,RA_1,RA_2,RA_3,RA_4分别表示用前述的各种可达性分析方法对 DR,LDR,DND,CGR 进行验证的结果.对 RA_1,RA_3 和 RA_4,生成单个抽象模型,这时,√表示该抽象模型中标错状态不可达,×表示该抽象模型中标错状态可达,T_1,T_3 和 T_4 表示相应的分析时间;对 RA_2,生成多个抽象模型,这时,√表示所有抽象模型的标错状态均不可

达,×表示存在标错状态可达的抽象模型,$\#N$ 为生成的实际抽象模型个数,$\#N_{\surd}$为标错状态不可达的抽象模型个数,$T_{2,\max}$ 和 $T_{2,\min}$ 分别表示单个抽象模型的最长分析时间和最短分析时间,T_2 表示对各个抽象模型进行可达性分析所需时间之和.在所有验证中,整型变量位数均为 3.对 RA_1,RA_2 和 RA_4 的验证中,全局数组 $\boldsymbol{O}$ 和 $\boldsymbol{D}$ 的长度均为 2;而在 RA_3 验证中,对抽象模型的中间状态 $s'_{i'}$,$t'_{j'}$ 判断 $s'_{i'} \sim_{\Gamma[entry]} t'_{j'}$, 使得抽象模型中 sto 操作数增加,从而导致作出正确可达性判断所需数组 $\boldsymbol{O}$ 的长度增长.在表 6 中,LEN 为 RA_3 验证中实际使用的 $\boldsymbol{O}$ 的长度,$\boldsymbol{D}$ 的长度仍为 2.

Table 6 Comparison of the precision of verification

表 6 验证精确性比较

用例	出处	DR[3]		RA_1	$T_1(10^{-3}\text{s})$	LDR[6]		RA_2	$\#N$ ($\#N_{\surd}$)	$T_{2,\max}/T_{2,\min}$ (10^{-3}s)	T_2 (10^{-3}s)
		Sf	Ty			Sf	Ty				
Avg	Sec.3.2[3]	√	√	√	5.37	√	√	√	2 (2)	5.46/1.80	7.26
AvgAttack	Sec.3.2[3]	×	×	×	9.93	×	×	×	2 (1)	9.75/2.30	12.05
Wallet	Sec.3.2[3]	√	√	√	8.65	√	√	√	2 (2)	8.77/3.81	12.58
WalletAttack	Sec.3.2[3]	×	×	×	12.24	×	×	×	3 (1)	12.31/2.80	24.63
ParityAttack	Sec.3.4[3]	×	×	×	7.30	×	×	×	2 (1)	7.21/3.21	10.42
Parity	Sec.3.4[3]	√	√	√	2.97	√	√	√	2 (2)	3.30/2.92	6.22
ParitySec	Sec.4[3]	√	×	√	4.94	√	×	√	2 (2)	4.93/3.16	8.09
pwUpdate	Sec.5[3]	√	√	√	50.38	√	√	√	4 (4)	55.57/2.69	78.88
WalletHash	Sec.5[3]	×	×	×	16.28	×	×	×	3 (1)	16.16/4.23	34.92
P1	Fig.3[5]	√	–	√	28.16	√	–	√	2 (2)	28.16/18.61	46.77
P2	Sec.1[6]	√	×	√	2.62	√	×	√	2 (2)	2.61/1.18	3.79
P3	Sec.1[6]	√	×	√	2.62	×	×	×	2 (1)	2.61/2.18	4.79
P4	Sec.3[6]	×	×	×	8.19	×	×	×	2 (1)	8.39/2.41	10.80
P5	Sec.3[6]	×	×	×	15.44	×	×	×	3 (1)	7.58/2.71	13.01
P6	Sec.3[6]	√	×	√	4.00	√	×	√	2 (2)	3.95/2.09	6.04
P7	Sec.4[6]	√	×	√	18.31	×	×	×	3 (2)	5.13/2.64	11.48
P8	Sec.4[6]	√	×	√	2.68	×	×	×	5 (4)	2.72/0.39	7.73
P9	Sec.5[6]	√	×	√	2.45	×	×	×	3 (1)	4.35/1.29	7.72
P10	Sec.2[9]	√	√	√	41.63	√	√	√	4 (4)	81.10/5.67	128.40
P11	Sec.2.1[2]	√	×	√	2.52	×	×	×	3 (2)	2.11/1.16	4.99

Table 6 Comparison of the precision of verification (Continued)

表 6 验证精确性比较(续)

用例	DND[10]		RA_3	LEN	$T_3(10^{-3}\text{s})$	CGR[11]		RA_4	$T_4(10^{-3}\text{s})$
	Sf	Ty				Sf	Ty		
Avg	√	√	√	2	40.13	√	√	√	5.37
AvgAttack	√	√	√	2	26.98	√	×	√	3.77
Wallet	√	√	√	2	20.57	√	×	√	8.65
WalletAttack	√	√	√	7	10.34	√	×	√	4.62
ParityAttack	√	√	√	2	7.63	√	×	√	3.22
Parity	√	√	√	2	6.88	√	×	√	2.97
ParitySec	√	√	√	2	6.61	√	×	√	2.88
pwUpdate	√	√	√	2	44.07	√	√	√	37.89
WalletHash	√	√	√	7	4.55	√	×	√	2.85
P1	×	–	×	4	228.90	×	–	×	60.53
P2	×	×	×	2	14.93	√	×	√	3.87
P3	×	×	×	2	5.93	×	×	×	7.19
P4	√	√	√	2	14.53	√	×	√	3.48
P5	×	×	×	2	11.23	×	×	×	8.55
P6	×	×	×	3	20.55	×	×	×	7.28
P7	√	×	√	2	9.71	√	×	√	4.14
P8	×	×	×	3	4.24	×	×	×	3.88
P9	√	×	√	3	11.84	√	×	√	1.59
P10	√	√	√	6	143.61	√	√	√	19.62
P11	×	×	×	2	3.89	×	×	×	4.91

程序 P1 的安全策略由于不是通过 declassify 显式表示,因而无法使用文献[3,6,10,11]中的类型系统.由于 P2 与 P3 使用 RA_1 方法的抽象结果相同,因而验证时间相等.与 RA_1 方法相比,文献[3]中 Sec.4 的类型系统具有保守性.保守性体现在以下几个方面:

(1) 严格拒绝形如 $l:=h$ 的语句,无论 h 是否已经在此前被 declassify 释放,如 P2,P3,P6,P8,P11.

(2) 通过类型规则中的 $U_1\cap D_2=\varnothing$保证在 declassify 中出现的变量不会在之前被更改,如 ParitySec 程序虽然实际泄露信息 parity(h)不多于允许泄露的信息 h,但仍被类型系统误判为不安全,再如 P7,P9.

(3) 与一般类型系统[19]类似,拒绝在高安全级上下文中对低安全级变量赋值,因而对 P7 和 P9 误判为不安全.

相对地,RA_1 方法则针对初态和终态下的变量间的语义等价性做判断,与定界释放定义一致,克服了上述前两种保守性;且基于语义的方法本身可以克服上述第 3 点保守性.文献[6]的类型系统在文献[3]的类型系统基础上要求 declassify 操作只能在低安全级上下文中进行,从而将 P5,P7,P9 判定为不安全.该类型系统未改进上述 3 点特性,因为局部定界释放本身相对定界释放更加严格,故导致类型系统的保守性只存在于对 ParitySec,P2,P6 的误判.由两种类型系统的判定结果还可以看出,虽然文献[3]的类型系统作为定界释放策略的实现机制,但实际上用其判定局部定界释放精确性更高.RA_2 方法对局部定界释放的验证亦具有精确性.将 RA_1 方法与 RA_2 方法比较,虽然 RA_2 方法中单个抽象模型一般来说比 RA_1 方法抽象模型更简单,但验证单个程序所需的抽象模型数由 1 个增长为多个,导致多数情况下总的验证时间更长.

在根据定界不披露性策略对用例进行安全性判断时,对用例中形如 l:=declassify(h)的语句看作 declassify(h) in {$l:=h$},当 declassify 出现在 if-else 语句的分支条件中时,将对应 declassify(h) in {c}的 c 取为整个 if-else 语句.定界不披露性在高安全级变量实际被释放之前而不是初态下判定变量等价性,使得策略不能防止洗钱攻击(laundering attack)[3],因而包含洗钱攻击的用例(P4,AvgAttack,WalletAttack,ParityAttack,WalletHash)均被定界不披露性定义为安全,这与定界释放和局部定界释放不同.实验结果说明,RA_3 方法可精确验证定界不披露性,而文献[10]的图 7、图 8 中,类型系统不再满足上述文献[3]中 Sec.4 的类型系统保守性中的前两条,是否拒绝 $l:=h$ 取决于语句所处的局部策略.若 $l:=h$ 不在局部策略 declassify(h) in {c}的 c 中,亦即所在的局部策略$\Gamma[i]$中 h 仍为高安全级,则类型系统拒绝 $l:=h$.

根据条件渐进释放定义对程序用例进行判定时,对 l:=declassify(e)标记的 flowspec 形式为 flow pre true & A(e) mod l,由于 R 中的前置条件ρ为 $true$,故这时根据条件渐进释放定义的判定结果与根据渐进释放的判定结果相同.从定义本身来说,条件渐进释放通过 flowspec 出现的位置规约 where 目标,通过 flowspec 中的前置条件ρ规约 when 目标,通过 flowspec 中的φ来规约 what 目标.但因为定义中 l:=declassify(e)的前置断言使用表达式 e 的当前值而非初始值,并没有规定在 e 中出现的高安全级变量在之前不能被更改,因而与定界不披露性类似,将包含洗钱攻击的用例定义为安全.但与文献[10]中类型系统不同的是,文献[11]的类型系统通过由 flowspec 中φ决定的集合 Pvars 来保证要被释放的高安全级变量不在其他位置被更改,从而实际实现了 what 目标并拒绝了洗钱攻击.相应地,这种类型规则的保守性使得程序 Wallet,Parity 和 ParitySec 被误判为不安全.与文献[3,6]的类型系统类似,文献[11]的类型系统严格拒绝 $l:=h$ 及高安全级上下文中对低安全级变量的赋值,故对 P2,P7,P9 产生误判.程序 P1 中,实际信息是通过 $l:=l+secret$ 语句释放的,若假定策略实际应用于 $l:=l+secret$,那么由于策略没有控制变量 $hash$ 的初始值的不确定性,因而导致在条件渐进释放策略下 P1 为不安全.由用例验证结果可知,RA_4 方法可精确验证条件渐进释放定义的安全性,对程序 Avg,Parity,Wallet 的抽象结果与 RA_1 方法相同因而验证时间相等.从实现机制上讲,文献[11]使用的局部自动验证与全局类型系统结合的方法,相对于 RA_4 的自动验证方法而言更为复杂,自合成只用于实现其局部自动验证.

由于文献[4]中 Sec.5.2 给出的类型系统限于函数式语言而无法直接应用于命令式语言,故无法在前述实验中判定其与第 1 节不严格无干扰性验证方法的相对精确性.文献[5]的方法使用 BLAST 对命令式语言上的不严格无干扰性进行验证,作为基于自动验证的方法,可与第 1 节对不严格无干扰性的验证方法进行验证效果比较.

文献[5]虽然给出了对程序进行自合成的方法,但没有给出自动的自合成实现,自合成需要手动完成.相比之下,本文方法实现了自动自合成并将存储-匹配操作内嵌于自动自合成过程中.实验使用 BLAST 版本 2.5.两种方法的验证结果如图 3 所示.图中横轴表示使用第 1 节方法时抽象模型中整型变量的位数,纵轴为验证时间.

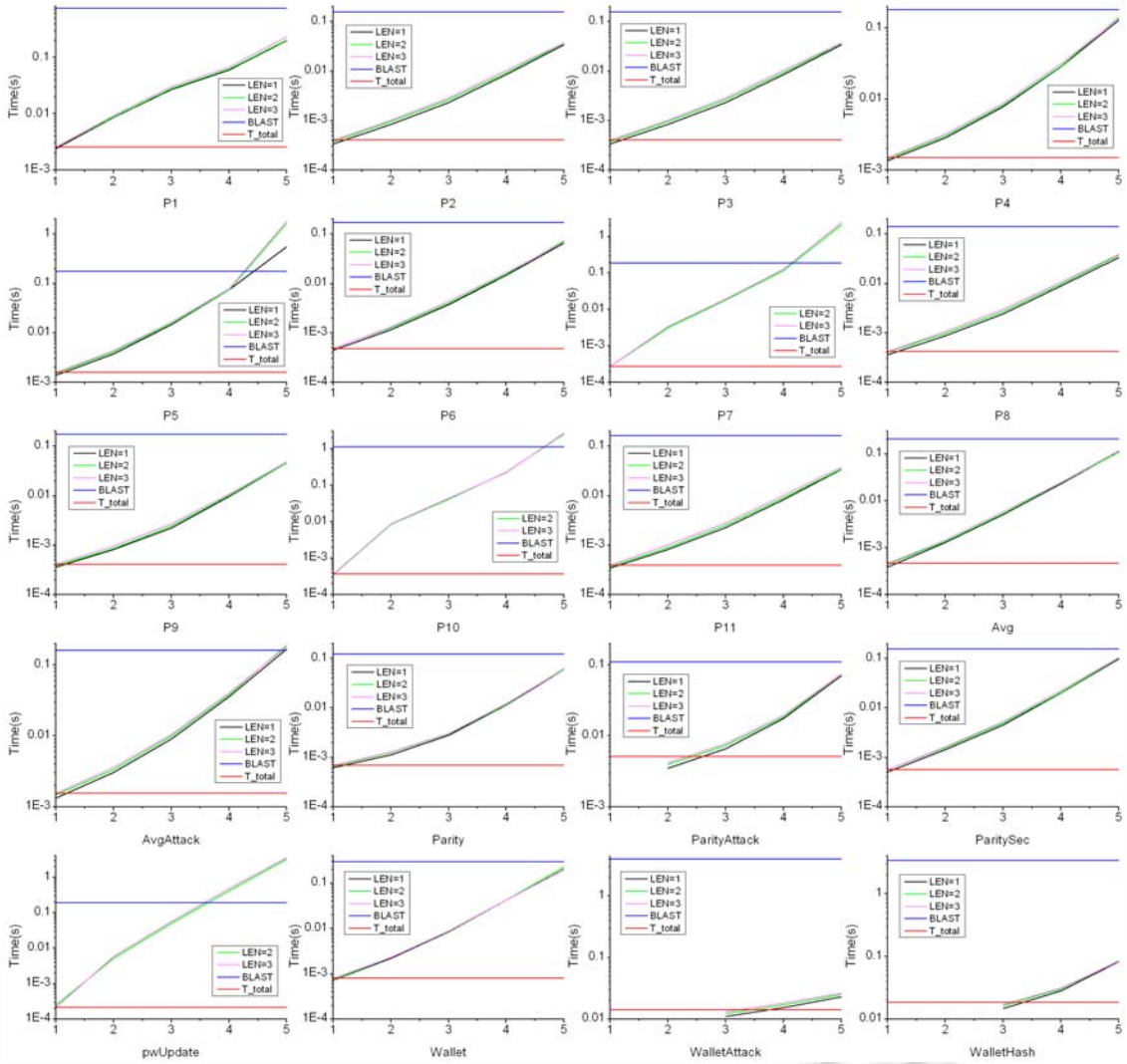

Fig.3 Comparison with Terauchi *et al.*[5] (using BLAST) on the effect of verification

图 3 与 Terauchi 等人[5]的方法(使用 BLAST)的验证效果比较

验证效果比较分为精确性和验证效率两个方面.

从精确性上讲,两种方法均为流敏感和上下文敏感的,对于本文中的所有例子,两种方法的精确性相同.当使用第 1 节方法时,过小的整型变量位数可能导致正误识(false positive),定义 $N_{\min}$ 表示为了避免正误识而要求整型变量位数所取的最小值,当整型变量所取的实际位数小于 $N_{\min}$ 时,则到达标错状态的路径由于符号抽象变得不可达,从而不能验证出原程序不安全(由图 3 可知,ParityAttack 的 $N_{\min}$ 为 2,WalletAttack 和 WalletHash 的 $N_{\min}$ 为 3,其他用例的 $N_{\min}$ 为 1).但由于对整型变量位数从 1 递增的多个抽象模型依此进行可达性分析总可以得到 $N_{\min}$,故正误识的存在不影响精确性.

从验证效率上讲,图 3 中 BLAST 为文献[5]方法的验证时间,由于抽象的实现方法不同,文献[5]的方法使用 BLAST 的验证时间对整型变量位数不敏感.图 3 中,LEN=k 表示当数组 $\boldsymbol{O}$ 和 $\boldsymbol{D}$ 长度均为 k 时第 1 节方法的验证时间.对于 P7,P10 和 pwUpdate,数组长度至少为 2 才能进行有效验证.由图 3 可知,数组长度 LEN 的增长只会导致验证时间的线性小幅增长,故不是验证开销的主要影响因素.而抽象模型的整型变量位数对第 1 节方法的验

证开销有实质影响,在整型变量实际位数大于等于 $N_{\min}$ 时,第 1 节方法的验证时间随整型变量实际位数的增长呈指数增长.对于可能引起正误识的用例,实际得到最终验证结果所需时间为 $T_{total}=\sum_{i=1}^{N_{\min}} t_i$,其中,t_i 即整型变量位数为 i 时第 1 节方法所用时间($t_1,t_2,...,t_{N_{\min}-1}$ 均得到正误识,$t_{N_{\min}}$ 首次将程序验证为不安全),而对 $N_{\min}$=1 的用例,$T_{total}=t_1$.由于 T_{total} 与不同的 LEN 值一一对应,因而可被看作对整型变量位数不敏感.当 *LEN*=3 时,各程序用例分别对应的 T_{total} 见图 3 中的 *T_total*.根据 T_{total} 与 BLAST 时间的比较可知,对于本文用例,第 1 节方法验证效率优于文献[5]的方法.

4 结束语

本文给出的机密消去机制可用于实现对定界释放、不严格无干扰性、局部定界释放、定界不披露性、条件渐进释放策略的支持.该方法比现有基于类型系统的方法具有更高的精确性,而在验证不严格无干扰性时,比其他基于自动验证的方法具有更高的效率.方法目前应用于命令式语言顺序程序.相对来说,其他一些相关工作则在类型系统的基础上对其他高级语言特性进行了一些讨论,如,Askarov 等人[9]在一般渐进释放定义的基础上,将机密消去与加密和密钥释放结合起来;Mantel 等人[12]将定界释放扩展到并发程序;Banerjee 等人[11]将条件渐进释放扩展应用于面向对象语言程序;Barthe 等人[10]将定界不披露性应用于字节码语言的一个子集.由于我们此前的工作[14]已在完整的字节码语言上讨论了终止不敏感无干扰性,因而未来的工作将在此基础上选择某些机密消去策略进行支持.对字节码的标准语义抽象,使得如何恢复 *l*:=declassify(*e*)中的表达式 *e* 变得困难,定界不披露性允许用变量预先替换被释放表达式的做法有助于解决这一问题.

基于类型系统的方法和基于自动验证的方法从应用环境上都属于静态分析方法,相对于静态分析的另一类方法是动态信息流监测[20−22].其中,Zi 等人[20]从抽象系统模型而不是语言特性的角度讨论隐式信息流和隐蔽信道;Tang 等人[21]讨论的污染分析实际上只包含显式信息流而不包含隐式信息流;Askarov 等人[22]提出的方法针对包含动态代码求值的命令式语言程序,首次对机密消去策略给出了形式化的动态监测机制支持.由于应用环境不同,本文未与此类动态机制进行验证的精确性比较,支持动态代码求值使得我们的方法易用于 Web 脚本语言,而动态监测的缺点在于,该方法会增加程序的运行时开销.

References:

[1] Goguen J, Meseguer J. Security policies and security models. In: Proc. of the IEEE Symp. on Security and Privacy. 1982. 11−20.

[2] Sabelfeld A, Sands D. Declassification: Dimensions and principles. Journal of Computer Security, 2009,17(5):517−548. [doi: 10.3233/JCS-2009-0352]

[3] Sabelfeld A, Myers A. A model for delimited information release. In: Kokichi F, Fumio M, Naoki Y, eds. Proc. of the ISSS 2003. LNCS 3233, Heidelberg: Springer-Verlag, 2004. 174−191. [doi: 10.1007/978-3-540-37621-7_9]

[4] Li P, Zdancewic S. Downgrading policies and relaxed noninterference. In: Palsberg J, Abadi M, eds. Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 2005. 158−170. [doi: 10.1145/1040305.1040319]

[5] Terauchi T, Aiken A. Secure information flow as a safety problem. In: Hankin C, Siveroni I, eds. Proc. of the SAS 2005. LNCS 3672, Heidelberg: Springer-Verlag, 2005. 352−367. [doi: 10.1007/11547662_24]

[6] Askarov A, Sabelfeld A. Localized delimited release: Combining the what and where dimensions of information release. In: Hicks M, ed. Proc. of the 2007 Workshop on Programming Languages and Analysis for Security. New York: ACM Press, 2007. 53−60. [doi: 10.1145/1255329.1255339]

[7] Mantel H, Sands D. Controlled declassification based on intransitive noninterference. In: Chin WN, ed. Proc. of the APLAS 2004. LNCS 3302, Heidelberg: Springer-Verlag, 2004. 129−145. [doi: 10.1007/978-3-540-30477-7_9]

[8] Matos A, Boudol G. On declassification and the non-disclosure policy. In: Proc. of the 18th IEEE Computer Security Foundations Workshop. IEEE Computer Society, 2005. 226−240. [doi: 10.1109/CSFW.2005.21]

[9] Askarov A, Sabelfeld A. Gradual release: Unifying declassification, encryption and key release policies. In: Proc. of the 2007 IEEE Symp. on Security and Privacy. IEEE Computer Society, 2007. 207−221. [doi: 10.1109/SP.2007.22]

[10] Barthe G, Cavadini S, Rezk T. Tractable enforcement of declassification policies. In: Proc. of the 21st IEEE Computer Security Foundations Symp. IEEE Computer Society, 2008. 83−97. [doi: 10.1109/CSF.2008.11]

[11] Banerjee A, Naumann D, Rosenberg S. Expressive declassification policies and modular static enforcement. In: Proc. of the 2008 IEEE Symp. on Security and Privacy. IEEE Computer Society, 2008. 339−353. [doi: 10.1109/SP.2008.20]

[12] Mantel H, Reinhard A. Controlling the what and where of declassification in language-based security. In: Nicola R, ed. Proc. of the ESOP 2007. LNCS 4421, Heidelberg: Springer-Verlag, 2007. 141−156. [doi: 10.1007/978-3-540-71316-6_11]

[13] Sun C, Tang LY, Chen Z. Secure information flow by model checking pushdown system. In: Proc. of the 2009 Symp. and Workshops on Ubiquitous, Autonomic and Trusted Computing. IEEE Computer Society, 2009. 586−591. [doi: 10.1109/UIC-ATC.2009.44]

[14] Sun C, Tang LY, Chen Z. Secure information flow in java via reachability analysis of pushdown system. In: Wang J, Chan WK, Kuo FC, eds. Proce. of the 10th Int'l Conf. on Quality Software. IEEE Computer Society, 2010. 142−150. [doi: 10.1109/QSIC.2010.50]

[15] Barthe G, D'Argenio P, Rezk T. Secure information flow by self-composition. In: Proc. of the 17th IEEE Computer Security Foundations Workshop. IEEE Computer Society, 2004. 100−114. [doi: 10.1109/CSFW.2004.17]

[16] BLAST: Berkeley lazy abstraction software verification tool. http://mtc.epfl.ch/software-tools/blast

[17] Holecek J, Suwimonteerabuth D, Schwoon S, Esparza J. Introduction to remopla. 2006. http://www.fmi.uni-stuttgart.de/szs/tools/moped/remopla-intro.pdf

[18] Keifer S, Schwoon S, Suwimonteerabuth D. Moped: A model-checker for pushdown systems. http://www.fmi.uni-stuttgart.de/szs/tools/moped/

[19] Volpano D, Irvine C, Smith G. A sound type system for secure flow analysis. Journal of Computer Security, 1996,4(2/3):167−188.

[20] Zi XC, Yao LH, Li L. A state-based approach to information flow analysis. Chinese Journal of Computers, 2006,29(8):1460−1467 (in Chinese with English abstract).

[21] Tang HP, Huang SG, Zhang L. Dynamic information flow analysis for vulnerability exploits detection. Computer Science, 2010, 37(7):148−151 (in Chinese with English abstract).

[22] Askarov A, Sabelfeld A. Tight enforcement of information-release policies for dynamic languages. In: Proc. of the 22nd IEEE Computer Security Foundations Symp. IEEE Computer Society, 2009. 43−59. [doi: 10.1109/CSF.2009.22]

附中文参考文献:

[20] 訾小超,姚立红,李斓.一种基于有限状态机的隐含信息流分析方法.计算机学报,2006,29(8):1460−1467.

[21] 唐和平,黄曙光,张亮.动态信息流分析的漏洞利用检测系统.计算机科学,2010,37(7):148−151.

孙聪(1982－),男,陕西兴平人,博士,讲师,主要研究领域为软件安全,程序分析.

陈钟(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络与信息安全,软件工程.

唐礼勇(1972－),男,博士,副研究员,主要研究领域为网络安全,系统软件.