一种多项式时间的路径敏感的污点分析方法

李佳静[1),2),3)] 王铁磊[2),3)] 韦 韬[2),3)] 凤旺森[2),4)] 邹 维[2),3)]

[1)](中国矿业大学(北京)机电与信息工程学院 北京 100083)

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

[3)](北京大学计算机科学技术研究所 北京 100871)

[4)](北京大学计算中心 北京 100871)

摘 要 提出了一种解决静态污点分析方法在进行路径敏感的分析时面临的路径爆炸的问题的方法.该方法将污点分析问题转化为加权下推自动机的广义下推后继问题,进一步利用污点数据在程序中的可达性,减少后续分析中需要精确执行的路径数.从而该方法能够以多项式的时间复杂度实现程序状态空间遍历,并能在发现程序违反安全策略时自动生成反例路径.设计实验使用该方法对击键记录行为进行了刻画,对恶意代码程序和合法软件进行了两组分析实验,并与现有的方法进行了对比分析.实验证明本文的方法可以有效地对具有较多分支的程序进行路径敏感的污点分析,同时具有较小的时间复杂度和空间复杂度

关键词 加权下推自动机;数据流分析;污点分析;恶意行为;击键记录

中图法分类号 TP309 **DOI 号**: 10.3724/SP.J.1016.2009.01845

A Polynomial Time Path-Sensitive Taint Analysis Method

LI Jia-Jing[1),2),3)] WANG Tie-Lei[2),3)] WEI Tao[2),3)] FENG Wang-Sen[2),4)] ZOU Wei[2),3)]

[1)](*School of Mechanical Electronic and Information Engineering, China University of Mining & Technology (Beijing), Beijing* 100083)

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

[3)](*Institute of Computer Science & Technology, Peking University, Beijing* 100871)

[4)](*Computer Center, Peking University, Beijing* 100871)

Abstract This paper proposes a method to solve the path explosion problem in path-sensitive taint analysis. The method transformed the taint analysis problem into a generalized pushdown successor problem by bringing the weighted pushdown system theory into taint analysis, so it could reduce the number of paths to be accurately executed. With this method, taint analysis can be performed in a polynomial time, and counter paths can be automatically generated. The authors designed experiments based on this method to model and detect keylogging behaviors. By comparing the authors' work with existing methods such as the raw traversal method and the full point-wise method, it is found to have exceptional efficiency in processing complex binary programs with a lot of branches, and has both a smaller time expense and a smaller space expense.

Keywords weighted pushdown system; dateflow analysis; taint analysis; malicious behavior; keylogger

1 引 言

污点分析方法目前已经在程序的安全性分析中得到了广泛的应用,包括程序安全漏洞分析[1-2]、恶意代码检测[3]和测试用例自动生成[4]等.

目前的污点分析方法的工作方式可以分为静态方式、动态方式和混合方式.静态方式的污点分析检

收稿日期:2009-04-14;最终修改稿收到日期:2009-08-07.本课题得到国家发改委信息安全专项(发改办高技[2007]2035)、教育部科技创新工程重大项目培育资金项目(707001)资助.**李佳静**,女,1979 年生,博士,讲师,研究方向为恶意代码分析、软件脆弱性分析.E-mail:ljj@cumtb.edu.cn.**王铁磊**,男,1985 年生,博士研究生,研究方向为软件脆弱性分析.**韦 韬**,男,1975 年生,博士,助理研究员,研究方向为反编译、软件脆弱性分析.**凤旺森**,男,1980 年生,博士,讲师,研究方向为近似算法与图论.**邹 维**,男,研究员,研究领域为网络与信息安全.

查程序的源代码或者可执行代码而不执行程序;动态方式通常在虚拟环境中监视程序的执行;混杂模式结合了具体执行和符号执行两种方式.静态方式的污点分析方法需要在源码中加入对安全类型的注释[2],或者面临路径爆炸问题[5].由于动态污点分析方式下只能观测到程序的有限次执行,并且它们的有效性依赖于测试集的完备性[3].为了解决这个问题,混合执行模式被提出来,例如 Ashcraft 等人实现的工具 EXE[4].混合执行模式提高了动态分析的路径覆盖率,它们的不足之处在于无法预知程序中数据的可达性,因此在选择程序执行路径时是盲目的.

本文提出了一种多项式时间的路径敏感的静态污点分析方法.该方法的基本思想为:根据污点数据在程序中的可达性,减少后续分析中需要精确执行的路径数.本文观察到污点分析的特殊性,即污点分析的数据流方程及其上的操作可以定义一个有界幂等半环.在此观察的基础上,本文将污点分析问题转换为加权下推自动机的广义下推后继问题,能够对程序进行路径敏感的污点分析,并且能以多项式的时间复杂度内遍历程序数据的状态空间.同时生成可能违反安全策略的路径,在这些路径上进行符号验证和路径满足性验证,可以降低静态分析的误报率,并且为后续的动态验证和特征提取等提供支持.

本文提出的静态污点分析方法已经在自主研制的 MalScope 系统中实现.在实验部分,使用该方法对击键记录行为进行了刻画,并针对恶意代码和合法软件进行了分析.与原始的路径遍历方法和逐点展开的方法进行了对比,分析结果表明,本文的方法能够有效地针对具有较多分支的程序进行污点分析,且具有较小的时间复杂度和空间复杂度.

本文第 2 节介绍加权下推自动机和可达性的概念;第 3 节阐述本文的基于加权下推自动机的污点分析方法;第 4 节给出本文的污点分析方法模型;第 5 节使用本文方法对恶意代码和合法程序的实验结果;第 6 节介绍污点分析的相关研究工作;最后在第 7 节给出本文的结论.

2 加权下推自动机和可达性

本文使用下推自动机为程序中的路径建模.加权下推自动机在下推自动机的基础上,为每个下推规则定义了一个有界幂等半环中的元素作为权重.本节给出下推自动机、有界幂等半环和加权下推自动机的概念.相关概念和术语参见文献[6].

下推自动机(PDS)表示为三元组$\mathcal{P}=(P,\Gamma,\Delta)$,其中 P 和 Γ 分别表示程序的状态和程序的栈符号的有限集合.$\mathcal{P}$中的一个配置表示为$\langle p,u\rangle$,其中$p\in P$,且 $u\in\Gamma^*$.Δ 为形式为$\langle p,\gamma\rangle\hookrightarrow_{\mathcal{P}}\langle p',u\rangle$ 的规则的有限集合,其中 $p,p'\in P,\gamma\in\Gamma$ 并且 $u\in\Gamma^*$.规则定义了$\mathcal{P}$中的配置之间的如下的变迁关系$\Rightarrow$:如果 $r=\langle p,\gamma\rangle\hookrightarrow_{\mathcal{P}}\langle p',u\rangle$,则对于任意的 $u'\in\Gamma^*$,$\langle p,\gamma u'\rangle\Rightarrow_{\mathcal{P}}\langle p',uu'\rangle$ 成立.在明确的上下文中脚标$\mathcal{P}$通常被省略.$\Rightarrow$ 的自反闭包记作$\Rightarrow^*$.给定一个配置集合 C,定义 $pre^*(C)=\{c'\mid\exists c\in C:c'\Rightarrow^*c\}$和 $post^*(C)=\{c'\mid\exists c\in C:c\Rightarrow^*c'\}$为 C 中的配置通过转换关系定义的后向可达等和前向可达集.

有界幂等半环(bounder idempotent semiring)为五元组$(D,\oplus,\otimes,\bar{0},\bar{1})$,其中 D 为一个集合,$\bar{0}$和$\bar{1}$是 D 中的元素,$\oplus$(联合操作)和$\otimes$(扩展操作)为 D 上的二元操作并且满足:

①$(D,\oplus)$为一个可交换的幺半群(monoid),$\bar{0}$是它的中性元(neutral element),并且$\oplus$操作是幂等的(即 $\forall a\in D,a\oplus a=a$);

②$(D,\otimes)$是一个幺半群,$\bar{1}$是它的中性元;

③$\otimes$操作对于$\oplus$操作是可分配的,即对于 $a,b,c\in D$,有$a\otimes(b\oplus c)=(a\otimes b)\oplus(a\otimes c)$和$(a\oplus b)\otimes c=(a\otimes c)\oplus(b\otimes c)$;

④$\bar{0}$是$\otimes$算符的零化子,即对所有的 $a\in D$,有$a\otimes\bar{0}=\bar{0}=\bar{0}\otimes a$;

⑤偏序关系$\sqsubseteq$定义为 $\forall a,b\in D,a\sqsubseteq b$,当且仅当 $a\oplus b=a$,D 中不存在无穷的降链(descending chain).

加权下推自动机(WPDS)为三元组$\mathcal{W}=(\mathcal{P},\mathcal{S},f)$,其中$\mathcal{P}$为下推自动机,$\mathcal{S}$为有界幂等半环,$f:\Delta\to D$ 是一个映射,为$\mathcal{P}$中的每条下推规则赋一个以有界幂等半环 D 中的元素作为权重.

令 $\Sigma\in\Delta^*$ 表示一个规则序列,使用 f 对 Σ 进行赋值,对于 $\Sigma=[r_1,\cdots,r_k]$,定义$v(\Sigma)=f(r_1)\otimes\cdots\otimes f(r_k)$.对于$\mathcal{P}$中的两个配置 c 和 c',用 $path(c,c')$表示从 c 到达 c'的所有规则序列$[r_1,\cdots,r_k]$.令 $C\subseteq P\times\Gamma^*$ 为一个正则配置集,广义下推后继问题(GPS)是为每个 $c\in P\times\Gamma^*$ 求解:

$$\delta(c)=\bigoplus\{v(\sigma)\mid\sigma\in path(c',c),\ c'\in C)\}.$$

3 基于加权下推自动机的污点分析方法

本文的方法可以简述为:将程序路径编码为

PDS,并将污点分析中的数据流方程定义为有界幂等半环,从而将污点分析问题转化为加权下推自动机中的 GPS 问题.本章首先给出待分析程序的语法定义,然后给出基于环境转换构建的污点分析中的数据流转换方程的有界幂等半环,最后给出本文建立 WPDS 的方法.

3.1　程序语法定义

为了检查程序中的信息流,本文定义带分析程序的语法如下:

expression $e::=n|val|x|f(e_1,\cdots)|e_1\sim e_2\}$;

statement $s::=e|x:=e|$ while(e) do $s|e:s_1,s_2,\cdots|$ goto $s|$return$\}$;

$val,x,f::=$string(value,variables and functions′ name).

其中,expression 为表达式;statement 为语句;n 为数值;val 为字符常量;x 为变量;~为表达式的二元操作;$f(e_1,\cdots)$为过程(函数)调用;语句 s 包括:表达式、赋值语句、返回语句、goto 语句、条件语句和循环语句.条件语句用来表示分支语句和开关语句,例如 if(e) then s_1; else s_2 表示为 $e:s_1,e_2$,即 s_1,s_2 是否执行取决于 e 的值;switch (e) case n_1 s_1; break;…; case n_m; break;表示为 $e:s_1\cdots s_m$.

3.2　污点半格和环境定义

本文定义污点半格为$(L^\top,\leq,\sqcap)$,其中 $L^\top=L\cup\top$,且 L 包含标准的污点分析中的两种安全类型 tainted 和 untainted,$\top$表示变量的安全类型未知.二元关系$\leq$指定了两个安全类型之间的偏序关系,该关系满足自反性和传递性.本文定义两个安全类型 $l_1\leq l_2$,当且仅当在安全类型为 l_2 的信息允许流入安全类型为 l_1 的变量中.对于两个安全类型 l_1,l_2,$l_1=l_2$ 当且仅当 $l_1\leq l_2$ 且 $l_2\leq l_1$.在本文的方法中定义 $tainted\leq untainted\leq\top$.以此偏序关系为基础,定义二元运算符$\sqcap$为污染格中的最大下界操作,同时满足结合律和交换律.在污染分析方法中定义当且仅当 $l_1\leq l_2$ 时,$l_1\sqcap l_2=l_1$.

令 V 表示程序中变量符号的有限集合,环境 $Env=(V\to L^\top)\cup\perp$表示 V 到 $L^\top$ 的映射,且一个特殊的元素$\perp$表示不可能的环境.定义环境 Env 上的操作如下:

(1) Env 的交操作定义如下:

$$env_1\sqcap env_2=\begin{cases}env_1, & env_2=\perp\\ env_2, & env_1=\perp\\ \lambda v.(env_1(v)\sqcap env_2(v)), & \text{其它}\end{cases}.$$

(2) env 的极大元记作 Ω,即 $\lambda v.\top$.

(3) 对于 $env\in Env$,$v\in V$,并且 $l\in L^\top$,表达式 $env[v\to l]$表示 v 的安全类型被映射为 l,其它变量 v'的安全类型被映射为 $env(v')$的环境.

3.3　污点分析的有界幂等半环

定义了环境之后,本文首先对程序中的路径进行表示,然后在此基础上将污点分析中的数据流分析问题转化为环境在路径上的转换,从而定义污点分析的有界幂等半环.

本文使用控制流图的超图 $G^*\in(N^*,E^*)$表示程序中的路径,其中包含多个控制流图 $G_1,G_2,\cdots$,(程序中的每个过程对应一个流图),使用 G_{main}表示程序的主过程,通常也为数据流分析的入口过程(也可以由用户指定某个过程作为分析的入口).每个控制流图有唯一的开始节点 e_p 和唯一的结束节点 x_p.流图中过程间调用使用两个节点(调用节点 ret_site和返回地址节点 ret_site)表示.流图中的其它节点表示过程内的语句和谓词.如图 1 中给出了 Rbot 程序代码的片段和它对应的超图.

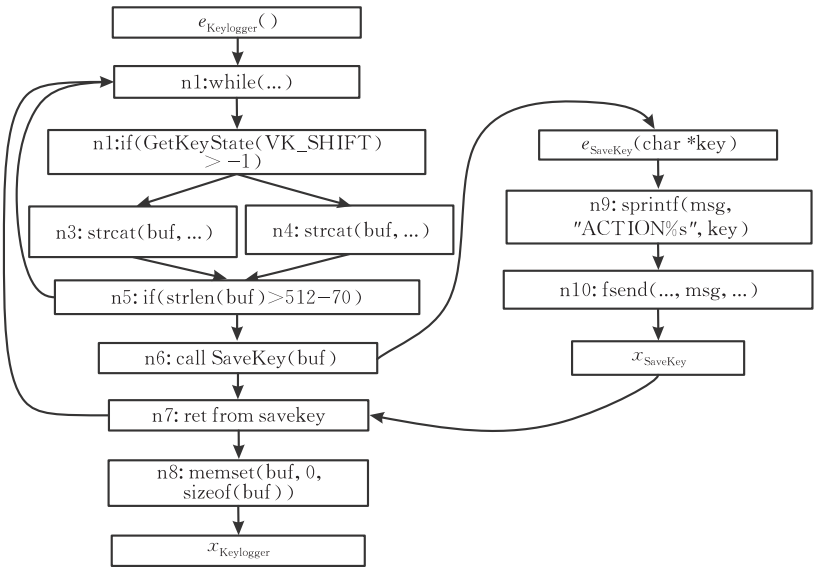
基于超图,本文为图中的每条边定义一个环境转换 $t:env\to env$ 来描述该边的起始节点中的数据操作对环境的影响.将执行 n 之前的环境称作参数环境,执行 n 之后的环境称作结果环境.首先引入符号 Λ 表示与参数环境无关的函数,使用 λ 演算对 t 的逐点表示 $R_t:(V\cup\{\Lambda\})\times(V\cup\{\Lambda\})\to(L^\top\to L^\top)$定义为

$$R_t(v',v)=\begin{cases}id, & v'=v=\Lambda & (1a)\\ \lambda l.t(\Omega)(v), & v'=\Lambda\wedge v\in V & (1b)\\ \lambda l.\top, & v',v\in V\wedge\forall l.t(\Omega[v'\to l])(v)=t(\Omega)(v) & (1c)\\ id, & v,v'\in V\wedge\forall l.t(\Omega[v'\to l])(v)=t(\Omega)(v)\wedge l & (1d)\\ \lambda l.\top, & l=\top & (1e)\\ \lambda l.t(\Omega[v'\to l])(v), & \text{其它} & (1f)\end{cases}$$

R_t 为每对符号(v',v)定义了一个微函数$R_t(v',v)$,表示它们之间的依赖关系.对于有$|V|$个变量的程序,每个 R_t 定义了$|V+1|^2$ 个微函数.在实际中,程序中的每条语句中只有少数变量的安全类型被重新定义,将数据流转换 t 中安全类型被改变的变量的集合定义为 $M(t)$.本文为每个 t 定义一个 R_t 的

```
void Keylogger(){
      char buf[512];
n1:   while(){
n2:     if(GetKeyState(VK_SHIFT>-1))
n3:           then strcat(buf,...);
n4:           else strcat(buf,...);
n5:     if(strlen(buf)>511-70){
n6, n7:       SaveKey(buf);
n8:           memset(buf, 0, sizeof(buf));
        }
      }
}
void SaveKey(char *key){
      char msg[512];
n9:  sprintf(msg, "NOTICE%s", key);
n10: send(..., msg, ...);
}
```

(a) Rbot代码片段



(b) 超图

图 1 Rbot 代码片段的超图

子函数 G_t，其对每个变量处理(1b)、(1d)、(1f)的情况. 则 R_t 包含两种微函数：(1) $G_t(\Lambda, v), v \in M(t)$；(2) $G_t(v', v), v', v \in M(t) \wedge G_t(v', v) \neq \lambda l.\top$. 例如对于语句 $v_1 = v_1 + v_2$. 该语句执行后 v_1 的安全类型依赖于 v_1 和 v_2 的安全类型，则定义该语句 G_t 为 $G_t(v_1, v_1) = \lambda l.l$ 和 $G_t(v_2, v_1) = \lambda l.l$. 对于给定的 G_t，它的解释 $[\![G_t]\!]$ 定义为如下的环境转换：

$$[\![G_t]\!](env) = \begin{cases} \bot, & env = \bot \\ \lambda v. \begin{cases} G_t(\Lambda, v)(\top) \sqcap (\underset{(v', v) \in domG_t}{\sqcap} G_t(v', v)(env(v'))), & (\Lambda, v) \in domG_t \\ \underset{(v', v) \in domG_t}{\sqcap} G_t(v', v)(env(v'))), & v \in M(t) \\ env(v), & \text{其它} \end{cases} \end{cases}.$$

若 $v \in M(t)$ 则在结果环境中 v 的安全类型被更新，否则 v 在结果环境中与参数环境中的安全类型一致. 对于每个 t，$t = [\![G_t]\!]$. 图 2 中显示了 G_t 与环境转换 $[\![G_t]\!]$ 的关系. 其中实线表示 G_t 中包含的微函数，虚线表示变量从参数环境到达结果环境，但是安全类型没有改变. 如图 2(a)所示，$M(t) = \varnothing$，则参数环境与结果环境一致，即 $[\![G_t]\!] = \lambda env.env$；图 2(b)中 $G_t(\Lambda, V_1) = \lambda l.\text{tainted}$，对应的环境转换为 $[\![G_t]\!] = \lambda env.env[v_1 \rightarrow \text{tainted}]$；图 2(c)中 $G_t(v_1, v_2) = \lambda l.l$，对应的环境转换为 $[\![G_t]\!] = \lambda env.env[v_2 \rightarrow env(v_1)]$；图 2(d)中 $G_t(v_1, v_1) = \lambda l.l$，$G_t(v_2, v_1) = \lambda l.l$，对应的环境转换为 $[\![G_t]\!] = \lambda env.env[v_1 \rightarrow env(v_1) \sqcap env(v_2)]$.

本文定义污点分析的有界幂等半环为 $\mathcal{S} = (D, \oplus, \otimes, \bar{0}, \bar{1})$，其中 D 为环境转换函数 t 的集合. 污点分析的有界幂等半环中的 $\oplus$ 操作定义如下：

$$t_1 \oplus t_2 = \lambda env.[\![G_{t1}]\!](env) \sqcap [\![G_{t2}]\!](env).$$

有界幂等半环中的 $\otimes$ 操作定义如下(其中 $\circ$ 为函数的复合操作)：

$$t_1 \otimes t_2 = [\![G_{t2}]\!] \circ [\![G_{t1}]\!].$$

有界幂等半环中的常量定义如下：

$$\bar{0} = \lambda env.\bot, \quad \bar{1} = \lambda env.env.$$

3.4 加权下推规则

本节首先给出将超图编码为 PDS 的方法，然后通过为 PDS 中的每条边定义一个污点有界幂等半环中的元素作为权重，建立 WPDS.

将待分析程序的超图编码为一个具有如下属性的 PDS $\mathcal{P} = (P, \Gamma, \Delta)$：$P$ 包含唯一的状态 p；每个栈符号 $\gamma_n \in \Gamma$ 对应超图中的点 n；每个下推规则 $\langle p, \gamma \rangle \hookrightarrow_{\mathcal{P}} \langle p', u \rangle \in \Delta$，按照下述方式对应于过程内的边、过程间调用和返回语句：

1. $\langle p, \gamma_n \rangle \rightarrow \langle p, \gamma_{n'} \rangle$ 为过程内从 n 到 n' 的边，表示程序的栈顶元素由 γ_n 变为 γ'_n，且栈的高度保持不变；例如图 1(b)中的 n_1 到 n_2 的边编码为 $\langle p, \gamma_{n1} \rangle \rightarrow \langle p, \gamma_{n2} \rangle$；

2. $\langle p, \gamma_n \rangle \rightarrow \langle p, \gamma_{n'} \gamma_{n''} \rangle$ 为从调用点 n 到 n'，且返回地址为 n'' 的函数调用，表示栈顶元素由 n 变为 n'，栈的高度增加一个单位；例如图 1(b)中的 n_6 的过程调用编码为 $\langle p, \gamma_{n6} \rangle \rightarrow$

$\langle p,\gamma_{n11}\gamma_{n7}\rangle$；

3. $\langle p,\gamma_n\rangle\rightarrow\langle p,\varepsilon\rangle$为返回节点，表示栈顶元素由 γ_n 变为空，栈的高度减少一个单位. 例如图 1(b)中的 x_{SaveKey} 节点编码为$\langle p,\gamma_{x_{\text{SaveKey}}}\rangle\rightarrow\langle p,\varepsilon\rangle$.

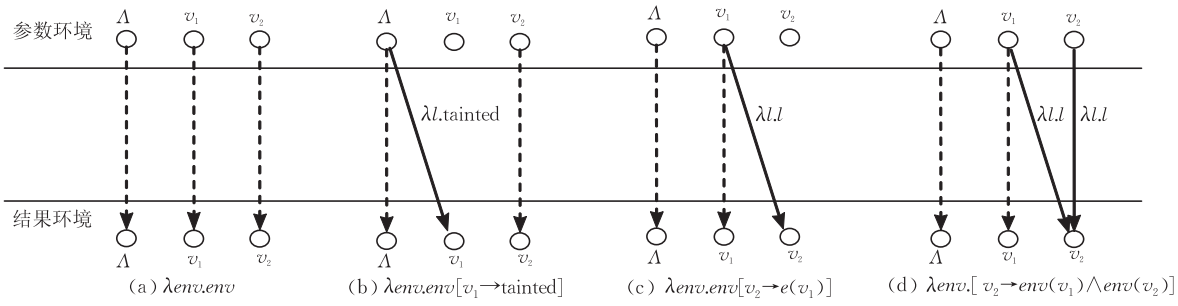


图 2　环境转换示意图

通过该编码方法将超图中的每条合法路径对应于 PDS 中的一条路径，反之亦然[6].

本文为 PDS 中的每个下推规则定义一个污点有界幂等半环中的元素作为权重，建立 WPDS. 假设全局变量能够到达程序中的任意节点，并且过程的局部变量能够到达本过程的任意节点. 在开始数据流分析时，程序的初始环境设置为 Ω；这里给出本文对特定的下推规则定义的权重，如表 1 所示.

表 1　加权下推规则

类型	γ_n	$\gamma_{n'}(\gamma_{n''})$	权重
R1	e_p	any	$[\![G_t(\Lambda,v)=\text{untainted},\ v\in globals\cup p.locals\cup p.args]\!],\ p=entry$ $[\![G_t(\Lambda,v)=\text{untainted},\ v\in p.locals]\!]$, otherwise
R2	SOURCE(T)	any	$[\![G_t(\Lambda,v)=\text{tainted},\ v\in T]\!]$
R3	SANITIZATION(T)	any	$[\![G_t(\Lambda,v)=\text{untainted},\ v\in T]\!]$
R4	PROPAGATION(S,T)	any	$[\![G_t(\Lambda,v)=\lambda l.l,\ v\in S,\ v'\in T]\!]$
R5	call_site	ret_site	$[\![G_t(\Lambda,v)=\lambda l.\top,\ v\in globals]\!]$
R6	call_site	e_p′(ret_site)	$[\![G_t(v',v)=\lambda l.l,\ v'=call.params[i],\ v=p'.args[i],$ $0\le i\le p'.args.len();\ G_t(\Lambda,v)=\lambda l.\top,\ v\in p.locals\cup p.args]\!]$
R7	x_p	ε	$[\![G_t(\Lambda,v)=\lambda l.\top,\ v\in p.locals\cup p.args]\!]$

下面分条解释：

(1) R1 表示过程的入口点的加权下推规则，默认情况下局部变量的安全类型被初始化为 untainted，若该过程为数据流分析的入口过程(例如 main)，则同时全局变量的安全类型被初始化为 untainted；

(2) R2 表示污点源 SOURCE(T)的加权下推规则，定义集合 T 中的存储地址的安全类型为 tainted. 本文支持如下的存储对象作为污点源：① 变量和内存偏移. 用户可以通过给出变量名和范围(例如全局变量或者一个过程的局部变量)指定一个内存区域为污染的；② 程序中的某个数据结构，例如可以指定一个程序的返回值为污染源；③ 从 I/O 流获得的数据，用户指定某种流类型读入的数据是污染的，例如网络、文件和键盘；

(3) R3 表示净化操作 SANITIZATION(T)的加权下推规则，使得 T 中的变量在执行后安全类型为 untainted；

(4) R4 表示污点传播操作 PROPAGATION(S, T)的加权下推规则，将操作的源操作数 S 的安全类型映射为产生数据 T 的安全类型. 其中产生数据 T 是程序中的存储对象且值被 S 修改的集合.

(5) R5 表示调用节点到返回节点的边的加权下推规则，全局变量无法通过调用节点到达返回节点，因此它们的安全类型为⊤；

(6) R6 表示过程间调用的加权下推规则，调用语句中使用的参数的安全属性被分别传播到被调用过程的参数；过程的局部变量在被调用过程中不可见，因此它们的安全类型为⊤；

(7) R7 表示过程出口节点的加权下推规则，局部变量和参数的生存周期结束，因此它们的安全类型为⊤.

(8) 以上规则中没有定义的规则的权重为$\bar{1}$.

R1～R8 的下推规则中只考虑了显示流的对变量的安全类型的影响，没有考虑隐式流[7]的影响. 显示流发生在变量间存在直接数据依赖关系的情况下，而隐式流发生在变量间通过控制流产生间接依赖关系的情况下. 例如语句：if ($v_3==0$) then $v_1:=v_2$ 当 $v_3==0$ 时执行 $v_1:=v_2$ 赋值，产生从 v_2

到 v_1 的显式流；但是不管 $v_3==0$ 是否成立，都产生从 v_3 到 v_1 的隐式流. 隐式流在条件语句和循环语句中出现，因此本文对这两种语句进行特殊处理：对于程序中的分支点，如果一个变量在该分支点和它的后必经节点间的某条语句中被更改，则为其增加一个微函数表示条件表达式中的变量通过隐式流对该变量的安全类型的影响. 对于循环语句采用相同的方法处理. 表 2 中的第 1、2 行分别给出了对条件语句和循环语句的处理方法.

表 2 条件语句、循环语句和检查点的处理方法

语句	处理方法
if exp then $c1$ else $c2$	$G_t=G_t\cup G_t(v',v)=\lambda l.l,\ v'\in var(exp),\ v\in M(t),\ t.from\in c1\cup c2$
while exp do $c1$	$G_t=G_t\cup G_t(v',v)=\lambda l.l,\ v'\in var(exp),\ v\in M(t),\ t.from\in c1$
$C(T,l)$	assert$(l\leq env(v),v\in T)$

定义了以上的概念之后，程序的安全策略定义为：程序中的所有执行序列在到达检查点 SINK(T,l) 时不会产生违反污点半格中≤操作所规定的信息流向关系，那么程序就是安全的. 本文支持两种检查点表示方法：(1) 使用变量名和范围来指定变量和内存的地址，指定一个过程的参数索引，也可以指定某个过程的入口或者出口，或者指定函数的返回值；(2) 指定待检查的指令的类型(例如系统调用或者跳转指令). 对检查点的处理方法如表 2 第 3 行所示.

4 基于加权下推自动机的污点分析系统模型

图 3 中给出了本文的静态污染分析方法的系统模型. 该模型包括 4 个主要部分：预处理器、下推规则生成器、模型检验器和验证器. 下面就它们的工作原理进行详细介绍.

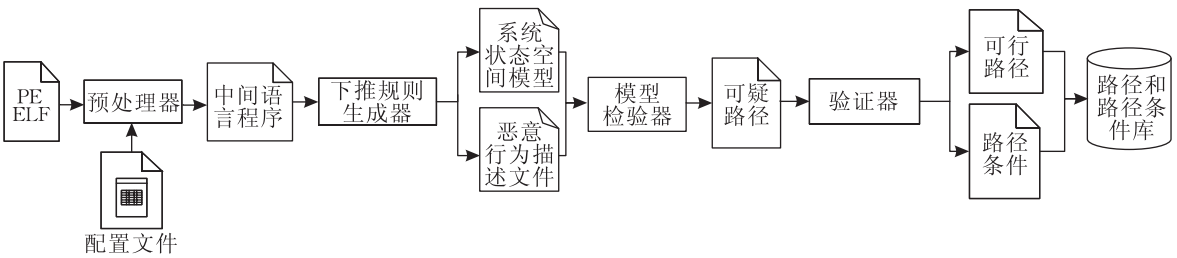


图 3 基于加权下推自动机的静态污点分析系统模型

4.1 预处理器

本文使用 IDA pro 5.0 完成反汇编工作. IDA pro 可以解析包括 PE 和 ELF 等多种可执行程序的格式，得到其中的汇编指令、数据说明、函数引入说明等信息.

汇编语言中存在很多不同于高级语言的特性，主要包括：(1) 结构化的控制结构信息丢失；(2) 变量没有明确命名，而是根据内存地址来索引以及存在隐含的变量标识和访问. 因此本文将通过控制结构识别、变量和操作数标识将其转换为 3.1 节定义的程序语法.

4.1.1 控制结构识别

在控制流分析阶段，本文针对分支语句和循环语句进行识别. 对于分支语句本文使用了级联分支网络的概念[8]. 该概念首先基于图论引入了相关控制结构的严格定义，然后通过包括扩展和收缩两个阶段划分分支语句的结构. 对于循环结构，本文使用了单次深度优先遍历的循环结构识别算法[9]. 该算法首先探讨了循环结构和深度优先遍历的关系，提出了一种基于栈结构的高效的单次遍历识别算法. 在这个过程中计算得到条件语句和循环语句的后必经节点.

4.1.2 变量和操作数标识

二进制代码中变量没有明确命名，而是根据内存地址来索引. 本文结合数据流分析实现了变量标识. 根据栈上的不同位置，将变量区分为过程的参数(例如 esp+4)、局部变量(例如 esp－$100h) 和临时变量，将栈地址空间中的位置集合作为变量符号表. 根据变量符号表，把汇编语言中的 push 和 pop 等命令转换为对栈指针 esp 以及当前地址上的变量的操作. 例如指令 push eax 被转换为两条指令 esp:=esp－4 和 tmp1:=eax，分别表示对栈指针和变量的操作. 将汇编指令翻译成高级语言中对应的操作，例如将 shl eax，2 翻译为 eax=eax*4.

汇编语言中对寄存器等的默认使用导致了一些隐含的数据操作，本文将这些操作数明确显示为等价的语句. 例如将 div 5h 翻译为 ah:=ax/5h；al:=ax% 5h. 在数据操作的同时，本文还考虑了符号寄存器的变化，例如 cmp eax，10h 被翻译为 zf:=eax－10h.

4.2　下推规则生成器

首先,下推规则生成器对需要进行数据流分析的过程个数进行精简.为每个过程计算它 call-from 集合,该集合中包含程序中所有调用该过程的过程名.然后本文采用如下方法计算有用过程的闭包:令 U 表示有用过程集合,在算法开始部分 $U=\varnothing$,然后将抽象语法树中包含了 SOURCE(T),SANITIZATION(T),SINK(T)和 PROPAGATION(S,T)操作的过程加入 U 中,再将 U 中的过程的 call-from 集合中过程加入 U 中,直到 U 不再改变.

之后建立程序的超图,该超图中只包含有用过程的流图.然后使用根据预处理器中识别的变量和 3.4 节中给出的方法,生成加权下推规则.本文将生成的下推规则保存在 XML 文件中,同时生成另外一个文件保存程序中所有的检查点.每个检查点包含需要检查的语句在超图中的节点号、需要检查的内存地址和它应该满足的安全类型.

4.3　模型检验器

基于以上的工作,本文可以将污点分析问题转换为 WPDS 的 GPS 问题,并使用模型检验求解该问题.

本文目前基于开源下推自动机库 WALi-2.3[①] 实现模型检验算法.WALi 基于 C++实现,其中的 WPDS 和 PA 都使用类实现.本文首先对类 WPDS 和 PA 等类定义为该库中定义的类的子类,然后解析规则生成器产生的 XML 文件,产生一个 WPDS 对象并且生成它的 $post^*$.最后对于每个检查点,检查在到达该节点指定的内存地址等的安全类型是否可能违反安全策略.在发现违法安全策略时,可以使用回溯算法自动生成反例路径.

在不考虑权重的情况下,令 $r=|\langle p,\gamma\rangle\rightarrow\langle p,\gamma'\gamma''\rangle|$ 表示在 WPDS 中至少包含一个形如 $\langle p,\gamma\rangle\rightarrow\langle p,\gamma'\gamma''\rangle$ 的规则的二元组 $\langle p,\gamma'\rangle$ 的个数,则模型检验算法的时间复杂度为 $O(|\Delta|\times r)$.即该算法的复杂度与程序中的分支数的多项式表达式.在考虑了权重的情况下,算法的复杂度为上述复杂度乘以一个不大于污点分析半环的最大降链的长度作为因数.在本文的方法中,假设程序中有 m 个变量,易知本文定义的污点分析半格的高度为 3,则污点分析半环的最大降链长度为 3^m.

4.4　验证器

路径生成器中生成的路径给出了路径上语句的执行顺序,本文基于符号执行和满足性验证在这些路径上进行验证.

4.4.1　符号执行

由于路径生成器已经给出了路径,包括过程调用和返回地址,符号执行只需对路径上的语句进行顺序处理.在初始的执行状态中,程序只包含入口函数的第一条指令地址,栈寄存器初始化为一个固定的具体值,其它的寄存器和内存位置被赋予符号值.随着符号执行,寄存器的内容(包括通用寄存器 eax,ebx,ecx,edx,esi,edi 和栈指针 ebp,esp)以及内存地址发生改变.本文使用 GiNaC[②] 符号系统实现符号执行.GiNaC 提供了一个 C++的符号计算相关操作的库,本文将语句的操作数定义为GiNaC中符号的子类,从而使用符号的计算来模拟符号过程中内存状态的改变.

4.4.2　可满足性验证

验证器验证当前路径条件是否可被满足,即当前路径是否为在真实执行中可能发生的路径.如果达检查时路径条件为可满足,则报告该路径,并将路径和路径条件存入数据库,以供包括路径特征提取和自动测试用例生成等后续分析.本文使用 STP 实现满足性验证.STP 实现了多数的算数操作(包括非线性操作、乘法、除法和求模运算)、按位的布尔操作等[10].本文只需将路径条件的表达式表示为位矢量,作为 STP 的输入,即可实现路径条件的约束求解.

5　实验与分析

本文选择对击键记录行为的分析来验证本文所提出的污点分析方法的有效性.这是因为击键记录是目前恶意软件最常用的信息窃取方法[③].这里击键记录是指用来实现监视和记录用户的击键动作的软件程序的方法.击键记录最常用的技术包括:(1)使用全局键盘钩子拦截系统按键消息;(2)循环查询键盘请求;(3)使用过滤驱动程序.据统计,95%的恶意代码中使用前两种技术实现击键记录.

击键记录的过程大致可以分为 3 个步骤:(1)使用以上 3 种方法收集系统中的击键信息;(2)将击键信息转换为易于人分析的形式,最常见的是转换为字符串;(3)将击键信息写入文件或者发送给攻击者.本文将恶意代码收集用户的击键信息并且将这些信息泄露给攻击者这种行为称作击键信息窃取

① http://www.cs.wisc.edu/wpis/wpds/
② http://www.ginac.de/
③ http://www.viruslist.com/en/analysis?pubid=204791931

行为，并将其定义如下：(1)使用击键记录技术收集用户的击键信息；(2)将收集到的击键信息泄露给攻击者.

为了刻画击键记录行为，在实验中 4 类特殊的操作分别定义如下：(1)污点源 SOURCE(T)，包括 GetKeyboardState、Get(Async)KeyState、GetKeyNameText 等 WinAPI 函数的返回值以及程序中通过 SetWindowsHook 自定义的全局键盘钩子函数中的参数；(2)净化操作 SANITIZATION(T)，包括标准 C 库中的 memset 以及汇编语言中的 XOR 操作等；(3)传播操作 PROPAGATION(S,T)，包括字符串拷贝和缓冲区操作等；(4)检查点 SINK(T,l)，包括标准 C 库中的 send、sendto、fprintf 和 WinAPI 函数 WriteFile 等写文件和网络通信函数的参数.

基于上述分析方法和系统模型，本文实现了一个静态污点分析系统 MalScope. 同时本文实现了另外两种方法与 MalScope 进行对比：(1)使用了本文的符号执行器和满足性验证器实现的 Kruegel 等人[5]提出的经典的路径遍历方法(称作 Traversal 方法)；(2)由于目前还没有多项式复杂度的路径敏感的静态污点分析方法，而 Sageiv 等人的逐点展开方法虽然是一种多项式复杂度的路径敏感的数据流分析方法，但是目前还没有应用到污点分析领域，并且逐点展开方式在定义微函数时定义的 R_t 使得生成了很多的冗余下推规则，因此本文使用 Sagiv 等人[11]的逐点展开方法(称作 FPW)，实现了一个类似 MalScope 的污点分析系统；FPW 的与 MalScope 的不同之处在于生成加权下推规则的方法不同. 为了验证本文方法在进行污点分析时的有效性和效率，本文设计了两组实验：(1)恶意代码中的击键信息窃取行为检测；(2)合法软件分析. 在进行以上实验的时候，本文记录了两种方法的运行时间和中间文件大小，以评价它们的时间复杂度和空间复杂度. 实验环境为 Opteron 2.6GHz CPU×2，8GB RAM，Linux 2.6.20-1.2962.fc6 系统.

5.1 恶意代码中的击键信息窃取行为检测

为了保证样本的真实性，本文使用自主研发的基于分布式蜜网 HoneyBow 的恶意代码捕获系统 Matrix[12]在 2008 年 9 月到 12 月间捕获的 3 种恶意代码，包括 IRCBot、KeyLogger、Rbot 的 7 个变种，共计 124 个样本. HoneyBow 系统已在全国 15 个省市自治区部署，获得了大量实际数据. 因此本文用来作对比分析的恶意代码都来自于实际的 Internet. 样本的分布可见表 3 中关于样本说明部分(样本、变种和数量 3 列)，其中 KeyLogger 是一个著名的以击键记录为主要功能的木马程序，它使用系统拦截的方式进行击键记录；IRCBot 和 Rbot 为典型的包含击键记录功能的僵尸程序，它们使用循环查询键盘请求的方式进行击键记录. 该样本集包含了实现击键记录的主要恶意代码类型(木马程序和僵尸程序)以及实现键盘记录的主要技术(系统拦截和循环键盘请求)以体现样本的代表性.

图 4 中给出本文使用 Traversal 方法对多个程序进行测试时，程序中分支个数与平均运行时间的关系图. 分析图中的数据可知其具有较显著的指数关系. Traversal 方法在程序分支数大于 60 时，在一天内无法停机甚至因为内存耗尽而崩溃. 因此本文认为该方法不适合对复杂程序进行处理，因此在之后只对 FPW 和 MalScope 两种方法进行对比.

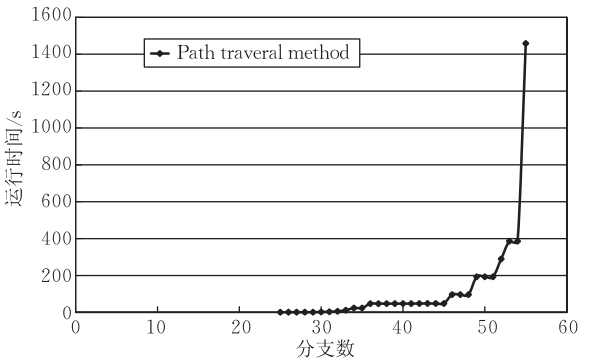


图 4 路径遍历方法的时间代价

在本次实验中 Malware 和 FPW 的检测结果相同，如表 3 所示. 检测结果中报告数指该方法判定为包含击键记录行为的样本数. 具体而言 MalScope 和 FPW 在 3 种恶意代码的 7 个变种中都报告了包含击键信息窃取行为的样本，共计 117 个. 经过人工审计，本次实验中 MALSCOPE、FPW 都没有发生漏报，这是因为它们采用了相同的行为刻画方法.

表 3 恶意代码检测结果

样本	变种	数量/个	FPW & MalScope 方法结果			
			报告数/个	写文件数/个	网络发送数/个	漏报数/个
IRCBot	az	10	7	6	7	0
	vn	1	1	0	1	0
Keylogger	lt	37	34	34	0	0
	r	3	3	3	0	0
Rbot	aea	27	26	26	26	0
	bng	22	22	22	22	0
	gen	24	24	21	24	0

表 4 中给出本次实验中 MalScope&FPW 的样本分析情况. 样本中的可达分支数表示样本中的平均可到达分支个数. 可到达分支个数是指从入口过

程(在本次实验中均为过程 start)开始,在下推自动机中能够到达的分支语句的数量.报告路径数给出报告的可疑路径总数;路径上分支数给出可疑路径上的平均分支个数;路径上过程数给出可疑路径上的平均过程个数.从该样本中可见,以 Rbot 为代表的僵尸程序的实现是比较复杂的,包含上千个分支和数个过程,使用类似 MalScope 的数据流自动分析工具进行辅助分析是必要的.

表 4　恶意样本及生成路径信息

样本	变种	可达分支数/个	报告路径数/个	路径上分支数/个	路径上过程数/个
IRCBot	az	3021	13	210	8
	vn	1222	1	53	8
Keylogger	lt	84	34	45	6
	r	132	3	35	8
Rbot	aea	3419	46	214	8
	bng	4064	44	231	9
	gen	3471	39	217	10

表 5 中给出了 MalScope 和 FPW 的时间代价,表 6 中给出了 MalScope 和 FPW 生成的下推规则的 XML 文件的大小.由于 FPW 中对于有 $|V|$ 个变量的程序,每个环境转换 R_t 定义了 $|V+1|^2$ 个微函数,从而定义了 $|V+1|^2$ 条规则.而 MalScope 中只对改变的变量定义微函数,并且对于每条语句值定义一条规则,因此具有较小的规则数目.从时间代价来讲,在本次实验中 MalScope 的效率高 1.5～3 倍;从空间代价来讲,MalScope 比 FPW 具有明显较小的空间代价,例如在本次实验中的 Rbot.gen 的样本集 MalScope 的空间代价是 FPW 的 1/56.

表 5　恶意代码检测的时间代价

样本	变种	预处理+规则生成空间代价/s		模型检验+验证空间代价/s	
		FPW	MalScope	FPW	MalScope
IRCBot	az	241	225	54	18
	vn	118	115	1	<1
Keylogger	lt	11	11	<1	<1
	r	27	27	<1	<1
Rbot	aea	337	310	102	34
	bng	466	411	140	91
	gen	331	299	82	29

表 6　恶意代码检测的空间代价

样本	变种	FPW 空间代价/MB	MalScope 空间代价/MB
IRCBot	az	143	<1
	vn	4	<1
Keylogger	lt	<1	<1
	r	<1	<1
Rbot	aea	241	4
	bng	436	8
	gen	224	4

5.2　合法软件分析

实验使用了 7 个合法软件作为分析样本,包括即时通信工具 HydralIRC 0.3.165、两个远程连接工具 Plink 0.60 和 VNC Viewer4、规则编辑器 MathType 5.0、文本编辑器 gvim71、Notepad2 2.1.19、在线播放软件 Miro 1.2.8.表 7 中给出了合法软件样本中的可达分支数.

表 7　合法软件的可达分支数

样本	可达分支数/个
HydralIRC 0.3.16	9428
gvim71	37250
MathType 5.0	5540
Miro 1.2.8	958
Notepad2 2.1.19	1456
VNC Viewer4	1212
Plink 0.60	1979

在本次实验中,两种方法都没有产生误报.这是因为合法软件没有使用本文刻画的击键记录行为将击键信息泄露出去.合法软件在处理用户的击键信息时,通常有两种情况:(1)处理用户在本进程内的输入,例如在 HydralIRC 0.3.165、MathType 5.0 等程序中使用 GetWindowText(A)、GetDlgItemText(A)等 API 处理用户在控件内的输入,这 7 个样本都没有使用全局键盘钩子函数处理用户击键信息.(2)在进程运行过程中查询键盘请求,接受用户的热键信息.例如 gvim71、Notepad2 2.1.19 中都使用 Get(Async)KeyState 查询 Ctrl 键是否被按下,在这种情况下合法软件不会将键信息写入文件或进行网络通信,而恶意代码会通过这些方式将击键信息泄露给攻击者.

在本次实验中两种方法都没有发现误报.可见本文的方法提供了一种对基于污点传播的行为的刻画手段,能够用来区分正常软件和恶意软件.表 8 中给出了两种方法的时间代价对比.表 9 中给出了 MalScope 和 FPW 方法生成的 XML 文件的大小.从时间代价来讲,在本次实验中MALSCOPE比

表 8　合法软件分析的时间代价

样本	预处理+规则生成时间代价/s		模型检验+验证时间代价/s	
	FPW	MalScope	FPW	MalScope
HydralIRC 0.3.16	1608	1604	6	5
gvim71	3162	2913	1150	157
MathType 5.0	1039	1042	3	1
Miro 1.2.8	52	53	<1	<1
Notepad2 2.1.19	679	665	8	<1
VNC Viewer4	320	334	13	4
Plink 0.60	336	315	2	1

FPW 的效率高 1.2～7 倍;从空间代价来讲,本次实验中 MalScope 比 FPW 的空间代价明显较小,例如对于 Notepad2 样本 MalScope 的空间代价是 FPW 的 1/17.

表 9 合法软件分析的空间代价

样本	FPW 空间代价/MB	MalScope 空间代价/MB
HydralIRC 0.3.16	8	8
gvim71	602	23
MathType 5.0	10	2
Miro 1.2.8	<1	<1
Notepad2 2.1.19	34	2
VNC Viewer4	48	2
Plink 0.60	6	2

5.3 讨 论

由上面的实验结果可知,MalScope 能够有效地解决路径爆炸问题,效率明显优于当前最经典的路径遍历方法以及同样具有多项式时间代价但仍未用于污点分析领域的 Sagiv 的方法.但需要指出的是,MalScope 在目前的实现中存在一些可能影响结果正确性的因素.

一方面,在使用 MalScope 对较大系统中的一个组件或者函数进行分析时,由于缺乏全局信息,如果在其它 MalScope 不可见的组件中,数据被污染或者净化,则本文的方法可能引起漏报或者误报.解决这种问题需要对其它的组件中的数据操作具有一些先验知识.

另一方面,MalScope 难以对事件触发和虚函数等的调用进行处理.在本文的下一步工作中将采用与动态分析相结合的方法,得到函数调用可能的上下文信息.

6 相关工作

污点分析在程序的安全性分析中得到了广泛的应用,包括软件漏洞分析,恶意代码检测等.这里讨论静态方式的污点分析方法.Shankar 等人在类型限制理论基础上,使用一个基于约束的类型推理机制来验证 C 语言中的信息流,检查在 C 程序中的格式化字符串缺陷[1].他们使用流不敏感的分析方法,变量的安全属性在程序中是不变的,因此会产生大量误报,而在本文的方法中为变量维护了类型环境,变量的安全类型随着程序中语句的执行发生改变.Foster 等人对 Shankar 的方法进行扩展,使用工具 CQual 在 Linux 内核中发现了一些未知的死锁问题[2],该方法需要在程序的源代码中加入对安全类型的注释,而本文的方法是针对可执行程序的,通过简单的配置即可进行污点分析.Kruegel 等人使用符号执行模拟数据在程序中的传播,当一个可加载的内核模块访问 rootkit 通常使用的内核地址时则报告它为可疑的 rootkitrootkit[5].该方法使用原始的路径遍历的方法,面临路径爆炸问题.而本文的方法利用污点数据的可达性为解决路径爆炸问题提供了一个思路.

7 结 论

本文提出了一种多项式时间的路径敏感的静态污点分析方法.该方法将污点分析的数据流转换方程定义为有界幂等半环,将污点分析问题转换为加权下推自动机的广义下推后继问题,从而可以使用模型检验方法求解.该方法能够对程序进行路径敏感的数据流分析,并且能在多项式时间内遍历程序的全局状态空间;该方法针对可执行程序,结合了多种数据流分析方法,包括符号执行和条件可满足性分析,可以求解包含特定行为的路径条件,为后续的动态验证等提供支持.本文提出的静态污点分析方法已经在自主研制的 MalScope 系统中实现.实验给出了它对真实的恶意代码和合法程序进行验证的结果.分析表明,该方法能够有效地进行路径敏感的污点分析.

参 考 文 献

[1] Shankar U, Talwar K, Foster J S, Wagner D. Detecting format string vulnerabilities with type qualifiers//Proceedings of the 10th USENIX Security Symposium. Washington, D. C, USA, 2001: 201-220

[2] Foster J, Terauchi T, Aiken A. Flow-sensitive type qualifiers//Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation. Berlin, Germany, 2002: 1-12

[3] Egele M, Kruegel C, Kirda E, Yin H, Song D. Dynamic spyware analysis//Proceedings of the 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference. Santa Clara, CA, 2007: 233-246

[4] Cadar C, Genesh V, Pawlowski P M et al. EXE: Automatically generating inputs of death. ACM Transactions on Information and System Security (TISSEC), 2008, 12(2): 10:1-10:38

[5] Kruegel C, Robertson W, Vigna G. Detecting kernel-level rootkits through binary analysis//Proceedings of the 20th Annual Computer Security Applications Conference. Tucson, AZ, USA, 2004: 91-100

[6] Reps T, Schwoon S, Jha S, Melski D. Weighted pushdown systems and their application to interprocedural dataflow analysis. Science of Computer Programming, 2005, 58(1-2): 206-263

[7] Denning E. A lattice model of secure information flow. Communications of the ACM, 1976, 19(5): 236-243

[8] Wei T, Mao J, Zou W et al. Structuring 2-way branches in binary executables//Proceedings of the 31st Annual International Computer Software and Applications Conference. Beijing, China, 2007: 115-118

[9] Wei T, Mao J, Zou W et al. A new algorithm for identifying loops in decompilation//Proceedings of the 14th International Static Analysis Symposium. Kongens Lyngby, Denmark, 2007: 170-183

[10] Ganesh V, Dill D L. A decision procedure for bit-vectors and arrays//Proceedings of the 18th Computer-Aided Verification Conference. Berlin, Germany, 2007: 81-94

[11] Sagiv M, Reps T, Horwitz S. Precise interprocedural dataflow analysis with applications to constant propagation. Theoretical Computer Science, 1996, 167(1-2): 131-170

[12] Zhuge J W, Han X H, Zhou Y L, Song C Y, Guo J P, Zou W. HoneyBow: An automated malware collection tool based on the high-interaction honeypot principle. Journal on Communications, 2007, 28(12): 8-13

LI Jia-Jing, born in 1979, Ph. D., lecturer. Her research interests include malware analysis and software vulnerability analysis.

WANG Tie-Lei, born in 1985, Ph. D. candidate. His research interests focus on software vulnerability analysis.

WEI Tao, born in 1975, Ph. D., assistant professor. His research interests include reverse engineering and software vulnerability analysis.

FENG Wang-Sen, born in 1980, Ph. D., lecturer. His research interests include approximation algorithm and graph theory.

ZOU Wei, born in 1964, professor. His research interests include Internet and information security.

Background

Behavior-based malware analysis is an important and hot issue in the area of information security at present. This paper focuses on improving the performance of behavior-based malware analysis methods. In this area, most current researches use static or dynamic taint-propagation technologies to detect backdoors, key loggers, network sniffers, and so on, which are good at detecting polymorphism and metamorphism, but have some obvious shortcomings: on one hand, the static methods suffer badly from the path explosion problem; and on the other hand, the dynamic methods usually fail to discover those behaviors that need special trigger conditions. This paper presents a method based on the theory of weighted pushdown system, to solve the path explosion problem for the static methods. By comparing it with other existing methods such as the raw traversal method and the full point-wise method, the work is found to have exceptional efficiency in processing complex binary programs with a lot of branches, and outperform them significantly in both time expense and space expense.

The institute of Computer Science & Technology of Peking University is a active group in the research of information security, and has developed a lot of advanced technologies. For example, the authors have deployed the malware capture system named Matrix in 15 provinces, and collected a large quantity of actual data from the internet, which also supplies experimental data for this paper. And the authors have proposed precise control structure normalization algorithms for malware reverse engineering, which make static analysis possible. This paper does a benefit work to improve the performance of static analysis, which is an important part of automatic malware analysis. This work is supported in part by a project from National Development and Reform Commission of China under grand No. [2007]2035, and an important project in scientific innovation from Ministry of Education of China under grand No. 707001. All of these projects aim to develop an effective mechanism to deal with complex malware such as those that have used polymorphism and metamorphism technologies.