ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.6, June 2010, pp.1426−1437
doi: 10.3724/SP.J.1001.2010.03507

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

一种隐式流敏感的木马间谍程序检测方法*

李佳静[1], 梁知音[2], 韦 韬[2], 邹 维[2+], 毛 剑[2]

[1](中国矿业大学(北京) 机电与信息工程学院,北京 100083)

[2](北京大学 计算机科学技术研究所,北京 100871)

Implicit-Flow-Sensitive Method for Detection of Trojan-Spy Programs

LI Jia-Jing[1], LIANG Zhi-Yin[2], WEI Tao[2], ZOU Wei[2+], MAO Jian[2]

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

[2](Institute of Computer Science and Technology, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: zouwei@icst.pku.edu.cn

Li JJ, Liang ZY, Wei T, Zou W, Mao J. Implicit-Flow-Sensitive method for detection of Trojan-spy programs. *Journal of Software*, 2010,21(6):1426−1437. http://www.jos.org.cn/1000-9825/3507.htm

Abstract: In this paper, a novel method is presented to solve these problems. This method processes the X86 executable programs statically, so it has a higher code coverage than dynamic methods. Besides, it employs a data flow analysis method to identify the jump targets for indirect jumps. It also utilizes optimized tainting mark rules based on the operation semantic of branch conditions. Experiments on 103 real malwares and 7 benign softwares show that the proposed method has the following advantages: For Trojan-spy program detection, it can reduce the false negatives caused by the explicit-flow-sensitive method, and it is effective in dealing with information steal behaviors triggered by some particular conditions. For benign program analysis, it can reduce most of the tainted branches that should be tracked in the original implicit-flow-sensitive method without optimization.

Key words: malware; behavior semantic; taint analysis; information theft; Trojan-spy program

摘 要: 提出了一种隐式流敏感的木马间谍程序检测方法.采用静态分析方式,具有更高的代码覆盖率;同时结合了数据流分析对间接跳转的目标进行计算;并且基于分支条件的操作语义,使用了针对木马间谍程序检测的改进的污点标记规则.应用该方法分析了 103 个真实的恶意代码样本和 7 个合法软件,并与现有方法进行了对比.实验结果表明,在进行木马间谍软件检测时该方法比显示流敏感的方法具有较低的漏报率,并且能够有效地发现需要特定条件触发的信息窃取行为.同时,该方法能够区分木马间谍程序和合法软件中的隐式流,显著消减对合法软件中的隐式流跟踪.

关键词: 恶意代码;行为语义;污点分析;信息窃取;木马间谍程序

中图法分类号: TP309 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2006AA01Z410, 2006AA01Z402 (国家高技术研究发展计划(863)); the Fundamental Research Funds for the Central Universities of China under Grant No.2009QJ15 (中央高校基本科研业务费); the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China under Grant No.707001 (国家教育部科技创新工程重大项目培育资金)

Received 2008-03-02; Accepted 2008-10-17; Published online 2010-04-12

随着网络游戏、网上银行和网上购物等互联网应用的日益流行,以信息窃取为目的的恶意代码已成为互联网用户面临的一个主要安全威胁.Symantec 分析报告指出,2006 年上半年收到报告最多的 50 个恶意代码中,48%的恶意代码可以帮助攻击者获得用户的敏感信息,这个数字在 2006 年下半年上升到 66%[1].其中,具有击键记录功能的恶意代码从 2003 年~2006 年间增加了超过 5 倍的数量,击键记录成为目前最常用的信息窃取技术[2].卡巴斯基实验室将具有击键记录功能的恶意代码定义为木马间谍程序(Trojan-Spy)[2],这类软件监视用户的击键活动,收集用户的击键信息并将这些信息泄露给攻击者,以进行网上银行或网络游戏的在线欺骗等.

为了降低信息窃取软件带来的威胁,许多检测工具应运而生.例如:AdAware[3],SpyBot[4]以及微软发布的反间谍软件 Windows Defender[5].这些检测工具主要使用基于语法特征的检测技术,但无法检测未知的或经过变形的恶意代码[6],因此出现了基于语义的恶意代码检测方法用于克服这种局限性.由于污点传播分析方法能够刻画程序对数据的非法使用和访问,因而成为目前基于语义的信息窃取软件检测方法中的研究热点,例如:Carnegie Mellon 大学的 Yin 等研究人员实现的通用恶意代码分析系统 Panorama[7].该方法在跟踪程序中的信息流时没有考虑隐式流[8]对数据安全类型的影响,因而可能产生漏报.除此之外,Egele 等人提出的对基于浏览器辅助对象(简称 BHO)机制的间谍软件的检测方法[9]以及 James 等人实现的通用动态污点分析框架 Dytan[10],这些方法在动态分析时对分支语句和它的后必经节点之间被执行的语句中的数据操作进行标记,为跟踪隐式流产生的数据传播提出了一个较好的思路.

木马间谍程序在实现时具有以下特点:

(1) 典型的样例大多结合远程控制技术实现(例如:Hupigon[11]);

(2) 通常采用隐式流方式传播其获取的用户击键信息(例如:Rbot[12]).

目前,尚没有专门针对木马间谍程序的检测方法,而现有的基于污点分析的方法在分析木马间谍程序时具有以下不足:

(1) 在结合远程控制技术实现的木马间谍程序中,击键记录行为只在收到控制端特定的命令时才被触发,现有方法采用的动态污点方式通常只能执行触发条件不成立的分支路径,而无法发现程序中的恶意行为;

(2) 现有方法在识别程序中的分支结构时,依赖于后必经节点的求解算法.对于使用间接跳转方式的分支语句,现有方法无法求解后必经节点从而失效;

(3) 木马间谍程序和合法软件在处理击键信息时均可产生隐式流,只有通过分支条件的操作语义才能对两者加以区分,但是现有的污点分析方法中(例如 Dyatan)只关注数据获得的途径(例如从网络获得),而不考虑数据的操作语义.

本文提出了一种隐式流敏感的木马间谍程序检测方法,以克服现有方法的不足.我们采用静态的隐式流敏感的污点分析方式,与动态分析方式相比具有更高的代码覆盖率,能够发现需要特定条件触发的击键记录行为.为此,我们扩展了现有的静态污点分析方法,提出了一套隐式流敏感的类型推导规则,能够刻画条件语句和循环语句对类型环境的影响;结合程序切片技术和模拟执行技术,对可执行程序的间接跳转语句的目标地址进行计算,提高了分支结构识别的准确性;基于分支条件的操作语义,提出了一个改进的污点标记规则,能够区分木马间谍程序和合法软件产生的隐式流,削减对合法软件中的隐式流跟踪.

本文提出的隐式流敏感的木马间谍程序检测方法已经在自主研制的 ISTA(implicit-flow-sensitive Trojan-spy analyzer)系统中实现.本文实验部分给出了它对真实的木马间谍程序和合法软件的分析结果,并与显示流敏感的方法以及没有优化的隐式流敏感方法进行了对比.分析结果表明,该方法能够跟踪采用隐式流进行的数据传播,在进行木马间谍软件检测时比显示流敏感的方法具有较低的漏报率,并且能够有效地发现需要特定条件触发的信息窃取行为.同时,该方法能够区分木马间谍程序和合法软件中的隐式流,显著削减对合法软件中的隐式流跟踪.

本文第 1 节介绍显示流和隐式流的概念以及击键信息窃取行为的刻画.第 2 节全面阐述隐式流敏感的木马间谍程序检测方法.第 3 节给出实验结果,并对实验结果进行分析和对比.第 4 节介绍恶意代码检测的相关研究

工作.第 5 节给出本文的结论.

1 知识准备

1.1 显示流和隐式流

Denning 指出,在分析程序的安全性时应该考虑程序中的两种信息流:显式流(explicit flow)和隐式流(implicit flow)[8].到达变量 y 的显式流发生在将变量 x 的信息直接传递给 y 时(例如赋值或者 I/O);到达变量 y 的隐式流发生在包含变量 x 的条件表达式的结果可以产生到达 y 的显式流时.例如,1 为 Yin 等人给出的击键记录代码片段[8],其中,x 中包含了用户的击键信息,x 安全类型为‘tainted’;通过从 x 到 y 的隐式流,y 中存储了与 x 中相同的数据,因此该语句执行后,y 应当具有与 x 相同的安全类型,即为‘tainted’.

```
switch(x){
case ‘a’: y:=‘a’; break; case ‘b’: y:=‘b’; break;…
}
```

Fig.1 A piece of code for keystroke recorder

图 1 击键记录代码片段

显示流敏感的污点分析方法(例如 panorama)的类型推导规则为:当一个赋值操作(包括存取操作和算术运算)的某个源操作数的安全类型是 tainted 时,该操作执行后目标操作数的安全类型为 tainted.对于图 1 中击键记录代码片段,字符常量 a(b…)的安全类型为 untainted,不满足推导规则,因此 y 的安全类型为 untainted.但根据前面的分析,y 的安全类型应为 tainted,即显示流敏感的污点分析方法在这种情况下会产生漏报.

1.2 击键信息窃取行为刻画

如前所述,木马间谍程序为具有击键记录功能的恶意代码.这里的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.击键记录最常用的实现技术包括:

(1) 使用 SetWindowsHook 等 API 建立全局键盘钩子(hook)拦截系统按键消息;

(2) 使用 Get(Async)KeyState 和 GetKeyboardState 等 API 循环查询键盘请求;

(3) 使用过滤驱动程序读取按键消息.据统计,95%的木马间谍程序使用前两种技术实现击键记录[2].

木马间谍程序的工作过程基本上可以分为以下 3 个环节:

(1) 通过特定的传播途径进入目标系统;

(2) 在目标系统中收集信息.使用上面提到的击键记录方法帮助攻击者提取用户应用程序中的击键动作信息;

(3) 泄露给攻击者或远端服务器.恶意代码窃取了用户击键信息之后,将这些信息记录在文件中或者发送给某个邮件地址等.

木马间谍程序的显著特征是,它收集用户的击键信息并且将这些信息泄露给攻击者.将这种行为称作击键信息窃取行为,该行为具有两个典型特征:(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.当一个程序包含上述两个特征时,就可以判定它为木马间谍程序.

2 隐式流敏感的木马间谍程序检测方法

本节将论述木马间谍程序检测方法,包括提出的隐式流敏感的类型推导和检查规则和以这些规则为基础的检测系统模型.

2.1 隐式流敏感的类型推导和检查规则

我们的方法中将程序抽象表示如下:

expression $e::=n|x|f(x)|e_1 \sim e_2$

statement s::=e|x:=e|s_1;s_2|e:s_1,…,s_n| while e do s_1

其中,expression 为表达式,statement 为语句,n 为常量,x 为变量,~为表达式的二元操作(例如“+”).$f(x)$为函数调用,其中包含 3 类不相交的特殊函数:

(1) 产生敏感信息的函数集合 T,程序通过这些函数收集用户的击键信息;

(2) 净化函数集合 S,程序通过这些函数消除数据的敏感性;

(3) 门函数集合 C,程序通过这些函数将信息泄露给攻击者.s::=e|x:=e 为简单语句,s::=s_1;s_2 为顺序执行语句,s::=e:s_1,…,s_n 为条件语句,即(多路)分支语句,while e do s_1 为循环语句.

现有的方法能够刻画显示流对类型环境的影响,但是没有考虑条件语句和循环语句中的隐式流对类型环境的影响.例如,Foster 的类型限制理论[13]中要求条件语句的不同分支具有相同的安全类型,也没有考虑到条件语句中的条件对汇聚点的影响,同时,Foster 的方法中没有处理循环语句.Strom 等人的类型状态中使用最大类型环境概念,即首先在每条执行路径上跟踪变量的安全类型的变化,然后再执行路径的汇聚点 r(例如循环的开始或者条件语句的结束),令 G_r 表示这些路径到达 r 时的类型环境集合,N 为程序中的变量集合,定义 $\Gamma=\vee G_r$ 作为最大的类型约束环境,即 $\forall x\in N, \Gamma(x)=\oplus_{\Gamma'\in G_r}\Gamma'(x)$.但是,Strom 的方法没有考虑条件语句中的条件对汇聚点的影响,对于循环语句使用相似的方法.我们在类型状态理论的基础上进行了扩展,提出了对条件语句和循环语句对类型环境影响的描述规则,形成以下的隐式流敏感的规则系统:

1. 现有方法对简单语句 s 和顺序语句 s_1;s_2 中的显示流对类型环境的显示影响用以下 7 条规则描述[14]:

(1) 常量规则:

$$\Gamma(n)=\text{untainted} \tag{1}$$

(2) 计算规则:

$$\Gamma(e_1\sim e_2)=\Gamma(e_1)\oplus\Gamma(e_2) \tag{2}$$

(3) 赋值规则:

$$\Gamma\square x:=e\Rightarrow\Gamma[x\mapsto\Gamma(e)] \tag{3}$$

(4) 产生规则:

$$\frac{f\in T}{\Gamma\square f(x)\Rightarrow\Gamma[x\mapsto\text{tainted}]} \tag{4}$$

(5) 净化规则:

$$\frac{f\in S}{\Gamma\square f(x)\Rightarrow\Gamma[x\mapsto\text{untainted}]} \tag{5}$$

(6) 串联规则:

$$\frac{\Gamma\square s_1\to\Gamma',\Gamma'\square s_2\to\Gamma''}{\Gamma\square s_1;s_2\Rightarrow\Gamma''} \tag{6}$$

(7) 检查规则:

$$\frac{f\in C,SATISFY(\Gamma,f,x)}{\Gamma\square f(x)\to\Gamma} \tag{7}$$

2. 条件语句 s=e:s_1,…,s_n 对类型环境的影响包括 3 个方面:

(a) 语句 $s_i(1\le i\le n)$通过显式流对类型环境的影响;

(b) 条件 e 通过隐式流对 s_i 产生的影响;

(c) s_1,…,s_n 汇聚点对类型环境的影响.

显式流对类型环境的影响在规则(1)~规则(7)中给出,下面针对(b)和(c)给出隐式流规则和汇聚规则.

令 M_i 表示在 s_i 中被显式流修改的变量集合,令 $I_i=N-M_i$.则对于 $\Gamma\square s_i\Rightarrow\Gamma_i$,可知 $\Gamma_i:=\Gamma_i(M_i\cup I_i)=\Gamma_iM_i\cup\Gamma_iI_i$.$\forall x\in M_i$,令 $\Gamma_i^*M_i(x):=\Gamma(e)\oplus\Gamma_iM_i(x)$.令 e:{s_i}表示 e 的值决定 s_i 是否被执行,则我们使用如下的隐式流规则描述 e 通过隐式流对 s_i 产生的影响:

$$\frac{\Gamma\square\ s_i \Rightarrow \Gamma_i}{\Gamma\square e:\{s_i\} \Rightarrow \Gamma_i^* M_i \cup \Gamma_i I_i} \tag{8}$$

$s_1,\ldots,s_n$ 在汇聚点对类型环境的影响可以用如下的汇聚规则来描述:

$$\frac{\Gamma\square\ s_1 \Rightarrow \Gamma_1,\ldots,\Gamma\square\ s_n \Rightarrow \Gamma_n}{\Gamma\square e:s_1,\ldots,s_n \Rightarrow (\Gamma_1^* M_1 \cup \Gamma_1 I_1)\vee\ldots\vee(\Gamma_n^* M_n \cup \Gamma_n I_n)} \tag{9}$$

3. 循环语句 s=while e do s_1 的循环体可能执行多次,我们采用迭代求得不动点(fixed point)的方式求解循环语句对类型环境的影响,并且加入循环条件产生的隐式流的影响.循环规则表示如下:

$$\frac{\Gamma_0 := \Gamma, \Gamma_0\square s \to \Gamma_1, \Gamma_1\square s \to \Gamma_2,\ldots; \Gamma'_0 := \Gamma, \Gamma'_1 := \Gamma'_0 \vee \Gamma_1, \Gamma'_2 := \Gamma'_1 \vee \Gamma_2,\ldots}{\Gamma\square \text{while } e \text{ do } s \Rightarrow \Gamma_{fixed}^* M \cup \Gamma_{fixed} I} \tag{10}$$

其中, $\Gamma_{fixed} := \Gamma'_i$,满足 $i \geq 1, \Gamma'_{i+1} := \Gamma'_i$ 且 $\Gamma'_i \neq \Gamma'_{i-1}$,M 和 I 与规则(8)中的定义相同.

2.2 隐式流敏感的类型推导和检查规则

本文的隐式流敏感的木马间谍程序检测方法可以简单地描述为:给定一个符合第 2.1 节定义的抽象程序,使用隐式流敏感的类型推导规则处理其中的语句序列,若程序中存在一条语句 s 使得在当前类型环境中检查 $SATISFY(\Gamma,f,x)$失败时,程序中包含不安全的信息流[15],则程序中存在击键信息窃取行为,并判定它为木马间谍程序.

图 2 给出了隐式流敏感的木马间谍程序检测方法的系统模型.该模型包括 4 个主要部分:反汇编器、汇编语言解析器、中间语言生成器和验证器.反汇编器将可执行文件生成汇编语言程序.汇编语言解析器对汇编语言程序进行词法解析和语法解析,生成它的抽象语法树(简称 AST).中间语言生成器完成两个功能:(1) 遍历程序的 AST,生成它的控制流图(简称 CFG);(2) 生成中间语言和符号表.验证器首先根据配置文件生成函数摘要,然后根据程序的 CFG 和符号表验证信息流的安全性,若程序中存在不安全的信息流则给出分析报告.

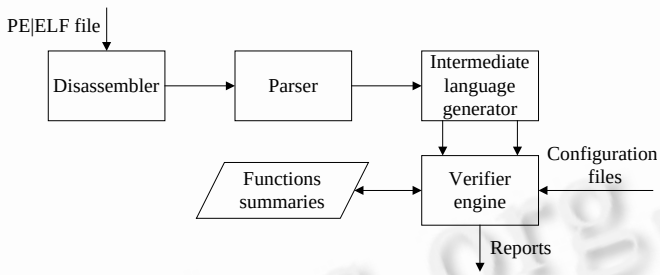

Fig.2 Architecture for implicit-flow-sensitive Trojan-spy programs detection method

图 2 隐式流敏感的木马间谍程序检测方法系统模型

本文使用 IDA pro 4.9 完成反汇编工作,得到其中的汇编指令、数据说明、函数引入说明等信息[16].汇编语言生成器根据 EBNF 规范将汇编语言生成 AST.下面详细介绍中间语言生成器和验证器的工作过程.

2.2.1 中间语言生成器

汇编语言中存在很多不同于高级语言的特性,主要包括:(1) 许多结构化的控制流信息丢失;(2) 存在隐含的变量标识和访问.因此将其转换为一种易于分析的中间语言,主要步骤包括控制流分析和中间语言生成.

2.2.1.1 控制流分析

汇编程序中存在的间接跳转使得反汇编器生成的控制流图出现中断,影响后续分析的准确性.为了得到连续的控制流图,我们实现了一种间接跳转语句分析算法,目前使用 IDA 插件完成.包括以下 4 个步骤:

(1) 标识需要处理的间接跳转的语句;

(2) 分析目标操作数的类型.如果是立即数、直接内存寻址或者函数参数的类型,则可以得到跳转目标或者是寄存器的值,进入第(4)步;如果是其他类型,进入第(3)步;

(3) 标识语句中用到的寄存器,对程序进行后向切片,直到找到对该寄存器的定义,或者找到该语句所在函

数的入口语句结束.如果后向切片到函数入口处还未得到该寄存器的定义语句,则认为该寄存器取任意值,否则以当前对寄存器的定义作为目标操作数进入第(2)步(第(2)步和第(3)步可能产生循环,但在对实际程序分析过程中,本算法在到达当前函数入口处时最多做 2~3 次后向切片);

(4) 判断后向切片得到的定义位置与目标间接跳转位置之间相关的寄存器是否被某些运算或操作改变.如果改变了,在切片得到的语句序列上模拟执行这种改变,直到到达间接跳转语句.根据间接跳转指令的模式,计算跳转表的所有可能索引值,或者调用的目标函数.目前,我们可以模拟加减乘除和移位运算.

对于间接调用分析我们采用相同的方法,对于其他控制结构,包括复合分支语句和循环语句的识别算法参见文献[17,18].

2.2.1.2 中间语言生成

汇编语言中的变量标识和数据操作与高级语言有很大差异,为此,我们将汇编语言转化为满足第 2.1 节中定义的抽象程序的中间语言,以在此基础上对程序进行验证.这一步主要完成以下两个方面的工作:

(1) 栈消除.汇编程序中包含对栈位置的隐含定义和使用.为了实现栈消除,我们结合数据流分析实现了栈模拟.根据栈上的不同位置,将变量区分为过程的参数(例如 esp+4)、本地变量(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,中间语言中把汇编语言中的 push 和 pop 等命令转换为对栈指针 esp 以及当前地址上变量的操作.例如在图 3 中,push eax 指令被转换为两条指令:esp:=esp–4 和 tmp1:=eax,分别表示对栈指针和变量的操作.

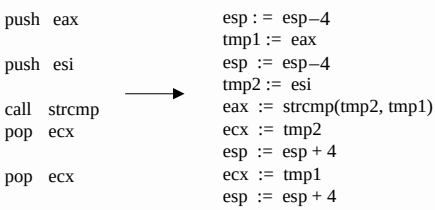

Fig.3 A sample for transforming ASM program to intermediate language

图 3 汇编语言转换为中间语言示例

(2) 汇编语言习语(idiom)转换.在汇编语言中对寄存器等的数据操作具有特别的定义,将这些操作在中间语言中明确表示.例如将汇编语言中,div esi 语句转换为以下语句 al:=ax/esi,ah:=ax%esi.;根据栈操作还原函数调用的参数,并且将函数返回值明确地赋给 eax.例如在图 3 的中间语言中,汇编语句的 call strcmp 被转换为 eax:=strcmp(tmp2,tmp1).

2.2.2 验证器

验证器遍历第 2.2.1 节生成的中间语言程序的 CFG,根据第 2.1 节定义的隐式流敏感的类型推导规则,静态验证程序中信息流的安全性.

首先,对于产生敏感数据的函数和净化函数,验证器根据它们对变量安全类型的修改情况生成它们的后置条件;对于门函数,验证器根据它们对变量安全类型的要求生成它的前置条件.

在遍历程序的 CFG 时,我们制定如下遍历规则:

(1) 每个基本块不能在它的前序基本块之前被遍历,除非该前序属于这个基本块为循环头的循环体内;

(2) 每个复合语句(循环、分支和多路分支)是一个连续的部分;

(3) 在满足上面两个条件下,优先遍历在二进制代码中先出现的基本块.

根据以上规则逐一对基本块进行分析.算法分为基本块输入合并、基本块内部执行,以及循环回绕迭代 3 个部分.

(1) 基本块输入合并部分:每个基本块开始执行时有一个状态表,记录每个变量的类型状态.在初始的执行状态中,只有入口函数(例如 main 函数)的第 1 个基本块,所有变量的安全类型都为 untainted.由于是按如前所述的次序执行,所以在基本块开始执行时,其正向前序基本块都已经执行完毕.若一个基本块有一个以上的正向前

序基本块,则它是一个分支汇聚点;若一个变量在多个正向前序基本块的状态表中出现,根据规则(9)计算变量的安全类型,更新状态表.

(2) 基本块内部执行部分:按照顺序逐一执行基本块内部的指令.在第 2.2.1 节的控制流分析过程中,为每个基本块记录了它所在分支结构的分支条件.在基本块内部执行时,根据规则(1)~规则(6)和规则(8)计算变量的安全类型,更改变量的状态表,从而在执行到基本块最后一步时形成输出状态表.同时还要根据规则(7)检验程序的安全状态,当发现程序中包含不安全的信息流时,给出分析报告.报告中给出门函数在程序中被调用的行号以及操作的类型(文件记录和网络传输).

(3) 循环回绕迭代部分:在基本块 b 执行完毕时,检查其后继基本块是否有包含 b 的循环头(这些循环头称为回边循环头).如果有,则记 b1 是满足条件的最外层的循环头,检查 b 的输出状态表和 b1 的输入状态表中是否有变量的类型状态发生变化,如果有,则根据规则(10)迭代回 b1 重新开始静态执行,直到状态表不再变化,即类型环境到达不动点(如果 b1 中没有符合条件的变量,则依次从外层向内层检查 b 的回边循环头).由于程序中变量的个数是有限的,同时污点格还具有有限的高度,而且我们的算法是单调递增的,可以证明该算法是能够停止的.在实验中,在到达不动点时最多进入循环体 2 次.

以图 1 中的代码为例,说明验证器的工作过程.将其表示为 s::=x:s_1,s_2,…,其中,s_1::=y:=‘a’,s_2::=y:=‘b’,依此类推.s 为条件语句,分别处理它的每个分支.例如 s_1,在其基本块输入部分$\Gamma(x)$=tainted,$\Gamma(y)$=untainted,Γ(‘a’)=untainted.在基本块内部执行部分,对于 s_1 中的显式流,根据规则(1)和规则(3)计算$\Gamma_1(y)=\Gamma$(‘a’)=untainted,即$\Gamma_1(y)$=untainted.而对于 x 对 s_1 产生的隐式流,$M_1=\{y\}$,$I_1=\{x\}$,根据规则(8)计算 $\Gamma_1^*(y)=\Gamma(x)\oplus\Gamma_1(y)$=tainted⊕tainted,即 $\Gamma_1^*(y)$ =tainted.在 s 中的语句都执行完后,下一条语句的开始部分根据规则(8)计算$\Gamma'(y)$=tainted⊕…⊕tainted.即在 s 执行后的状态表中,x 的安全类型保持不变,y 的安全类型改变为 tainted.

3 实验分析

第 2 节给出了本文的方法模型,但在实现中还有很多具体问题需要考虑.本节给出对这些具体问题的解决思路,然后给出实验结果.

实验中对特殊函数集合分别定义如下:

(1) 产生敏感数据的函数集合 T,包括 GetKeyboardState,Get(Async)KeyState,GetKeyNameText 等 WinAPI 函数,以及程序中通过 SetWindowsHook 自定义的全局键盘钩子函数等;

(2) 净化函数集合 S,包括标准 C 库中的 memset 等;

(3) 门函数集合 C,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.

这与第 2.1 节中 3 类特殊函数的定义是对应的.

在木马间谍程序中,通过隐式流进行数据传播主要有两种方式:

(1) 通过 SetWindowsHook 建立全局键盘钩子或使用过滤驱动程序获得键盘输入信息的缓冲区,使用如图 1 所示的多路分支语句,将击键信息转换为对应的字符;

(2) 通过调用 Get(Async)KeyState 等 API 循环查询键盘请求,对查询的结果进行判断,将击键信息转换为对应的字符.

对于全局键盘钩子的方式,由于其采用事件触发模式,在我们的实现中,根据 SetWindowsHook 中指定的处理函数地址,将程序中对 SetWindowsHook 的调用替换为对该处理函数的调用,在此上下文中分析消息处理函数.这种方式下,木马间谍程序和合法软件获取击键信息的途径不同,因此不需要区分木马间谍程序和合法软件的隐式流.对于过滤驱动程序,我们的方法目前没有对其进行处理.

对于循环查询键盘请求方式,由于合法软件会使用 Get(Async)KeyState 等 API 在进程运行中处理用户的热键信息,这与木马间谍程序的查询键盘请求方式具有相似之处,因此需要对两者进行区分.经过对大量合法软件的分析,我们发现合法软件处理的热键信息有两个特点:(1) 调用 Get(Async)KeyState 查询的通常为不可见字符

(例如 Ctrl,Alt,F1 等键),对可见字符的查询通常伴随着对不可见字符的查询;(2) 调用 Get(Async)KeyState 的参数通常为常量.相比而言,木马间谍程序中处理的击键信息具有关注可见字符;调用 Get(Async)KeyState 的参数可能为变量的特点.我们制订了如下改进的污点标记规则:

(1) 当 Get(Async)KeyState 的参数为对可见字符(包括字母 a~z,数字 0~9 等)的查询,或者参数为变量时,它的返回值的安全类型为 tainted;

(2) 为对 Ctrl,Alt 和 Win 键的查询结果增加一个特殊属性 clean,当复合分支中的某个条件的属性为 clean 时,该复合分支条件被净化为 untainted,以区分合法软件中使用组合键作为热键的情况.

对于可执行程序中复合分支条件的概念和识别算法见文献[17].改进的污点标记方法在第 2.2.2 节基本块内部执行部分实现.

基于上述检测方法和系统模型,我们实现了一套隐式流敏感的木马间谍程序检测工具 ISTA.根据恶意代码检测方法的两个评测指标即漏报率和误报率,设计了两方面的实验进行对比分析:

(1) 评测在检测木马间谍程序时的漏报率,同时评测发现需要特定条件触发的信息窃取行为的能力;

(2) 评测在分析合法软件时的误报率.

在两组实验中,我们对比了显示流敏感的方法(以下简称 ESM),加入了改进规则的显示流敏感方法(以下简称 AESM),不包含改进规则的隐式流敏感的方法(以下简称 ISM)和 ISTA 的分析效果.测试环境为 AMD Opteron 2.6Ghz×2CPU,8G RAM,Linux 2.6.20-1.2962.fc6 系统.

3.1 实验1:木马间谍程序检测和漏报率

本实验将评测 ESM,AESM,ISM 和 ISTA 在检测木马间谍程序时的漏报率情况,以及发现只在特定条件下发生的信息窃取行为的能力.

我们使用自主研发的基于分布式蜜网 HoneyBow 的恶意代码捕获系统[19]在 2007 年 8 月~10 月间捕获的 3 种恶意代码,包括 IRCBot[20],KeyLogger[21],Rbot[12]的 7 个变种,共计 103 个样本.样本的分布见表 1 中关于样本说明部分(Sample,Variation 和 Count 这 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,用于远程控制被害主机.

Table 1 ESM/AESM in Trojan-spy programs detection

表 1 ESM/AESM 对木马间谍程序的检测结果

Sample	Variation	Count	ESM/AESM			
			Reports	File writing	Network sending	False negatives
IRCBot	az	8	0	0	0	5
	vn	1	0	0	0	1
KeyLogger	lt	25	22	22	0	0
	r	3	3	3	0	0
Rbot	aea	4	0	0	0	29
	bng	17	0	0	0	15
	gen	15	0	0	0	12

在本次实验中,ESM 和 AESM 的检测结果相同,表 1 给出它们的检测结果.这是因为改进的污点标记方法针对分支条件进行优化,而显示流分析方法中不处理分支条件,因此改进的方法对分析结果没有影响.ISM 和 ISTA 对木马间谍程序的检测结果相同,结果在表 2 中给出.这是因为 ISTA 改进的污点标记规则刻画了木马间谍程序的本质,虽然比 ISM 中的规则增加了约束,但是仍然能够检测到木马间谍程.即改进的污点标记方法没有增加检测方法的漏报率.检测结果中,报告数(reports)是指该方法判定为木马间谍程序的样本数,文件记录(file writing)是指将击键信息写入文件的样本数,网络传送(network sending)是指将击键信息通过网络传送的样本数.本次实验中,ISM/ISTA 的漏报率为 0%,而 ESM/AESM 的方法的漏报率为 71.3%.具体而言,ISM/ISTA 在 3 种恶意代码的 7 个变种中都报告了木马间谍程序样本,共计 87 个.经过人工审计,没有发现漏报.而 EMS/AESM 只报告了 KeyLogger.lt 和 KeyLogger.r 中共 25 个木马间谍程序样本,无法发现其他两种恶意代码中的木马间谍程序样本.

经过分析发现,这是因为 IRCBot 和 Rbot 都采用隐式流方式实现击键信息窃取.

对于需要特定条件触发的信息窃取行为的检测,IRCBot 和 Rbot 都使用远程控制的方法进行击键信息窃取,这只有当被控机器收到控制端的特定命令时才会发生.ISTA 对 IRCBot 和 Rbot 中 62 个使用远程控制实现击键信息窃取的样本检测中,没有发现漏报.

Table 2 Results of ISM/ISTA in Trojan-spy programs detection

表 2 使用 ISM/ISTA 方法对木马间谍程序的检测结果

Sample	Variation	Count	ISM/ISTA			
			Reports	File writing	Network sending	False negatives
IRCBot	az	8	5	3	5	0
	vn	1	1	0	1	0
KeyLogger	lt	25	22	22	0	0
	r	3	3	3	0	0
Rbot	aea	34	29	29	29	0
	bng	17	15	15	15	0
	gen	15	12	9	12	0

通过对样本进行分析,本次实验中使用的 103 个样本都使用了间接跳转语句,包括跳转表和函数指针.图 4 给出了在分析木马间谍程序时需要进行隐式流分析的分支条件个数(在本次实验中,ISM 和 ISTA 分析的分支条件个数相同,分支个数为该变种的多个样本中需要跟踪的分支个数之和).图 5 给出了 ESM,AESM,ISM,ISTA 这 4 种方法对本次实验使用的 3 种恶意代码的 7 个变种进行分析的运行性能(运行时间为该变种的多个样本的平均运行时间),其中,ISTA 和 ESM 性能最大差距值为 8.6%(Rbot.gen 样本集).

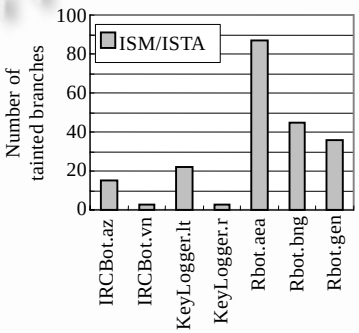

Fig.4 Numbers of tainted branches in Trojan-spy program detection

图 4 木马间谍程序检测时污点分支数

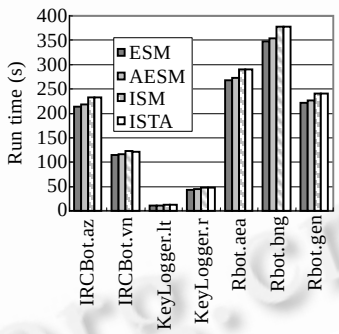

Fig.5 Performance in Trojan-spy program detection

图 5 木马间谍程序检测的运行性能

3.2 实验2:合法软件分析和误报率

本实验评测 ESM,AESM,ISM 和 ISTA 对分析合法软件时的误报率情况.实验使用了 7 个包含处理用户的击键信息功能的常见软件作为分析样本,包括两个即时通信工具 Skype3.0.4 和 msnmsgr 8.1,两个远程连接工具 SecureCRT5.0.3 和 Xmanager 2.0,以及规则编辑器 MathType 5.0,Goole 桌面 GoogleDesktop 5.1 和金山词霸 xdict 10.0.0.1 程序.

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 Skype3.0.4, msnmsge8.1,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,7 个样本中都使用 Get(Async)KeyState 查询‘Ctrl’键是否被按下,在这种情况下,我们使用前面改进的污点标记规则来进行区分.

在本次实验中,ESM,AESM,ISM 和 ISTA 这 4 种方法都没有发现误报.改进的污点标记方法没有增加检测的误报率.ESM 和 AESM 不对程序进行隐式流跟踪.图 6 给出了使用 ISM 和 ISTA 两种方法进行隐式流跟踪的分

支个数,ISTA 对合法软件进行隐式流跟踪的分支数最多为 2 个,而 ISM 跟踪的分支数平均为 12 个.其中,ISTA 对合法软件进行隐式流跟踪是因为程序中使用变量作为 Get(Async)KeyState 的参数,例如,Skype3.0.4 中将自定义的过程返回值作为 Get(Async)KeyState 的参数.可见,ISTA 能够显著削减对合法软件的隐式流跟踪.

图 7 给出了 4 种方法对合法软件分析时的运行性能,其中,Skype3.0.4 的可执行文件大小为 24.3MB,ISTA 和 ESM 算法的性能差距为 10.3%.

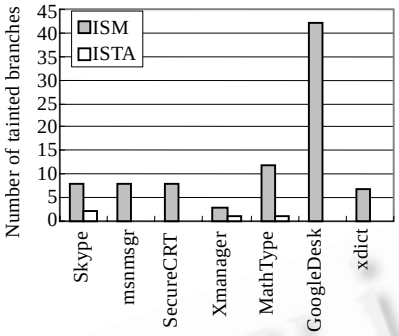

Fig.6 Numbers of tainted branches in benign software analysis

图 6 对合法软件分析时进行隐式流跟踪的分支个数

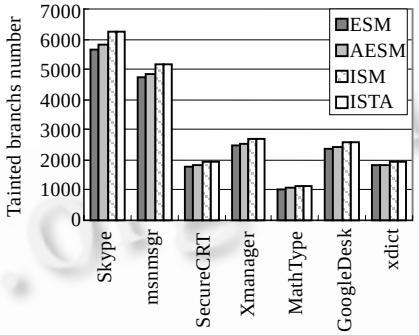

Fig.7 Performance in benign software analysis

图 7 对合法软件分析时的运行性能

4 相关研究工作

传统的恶意代码检测工具主要使用基于语法特征的检测技术,无法检测未知的或经过变形的恶意代码[6],因此出现了基于语义的恶意代码检测方法来克服这种局限性.由于污点分析方法能够跟踪数据在程序中传播的过程,刻画对数据非法使用的恶意行为,因而成为近年来恶意代码分析方法研究的热点.

Kirda 等人首先提出了对基于 BHO 机制的间谍软件的行为刻画和识别,他们首先使用动态分析找到需要分析组件的入口,然后静态分析提取这些组件的控制流图.若控制流图中调用了某些可疑函数,则该组件可能是间谍软件[6],但该方法无法识别间谍软件窃取的数据信息,而且会产生误报.Yin 等人使用动态污点分析在系统范围内检查信息窃取行为,可以检测网络监听程序、后门程序和基于 BHO 的间谍程序等恶意软件[7].该方法在处理信息流时没有考虑到由隐式流产生的数据传播.

由于不能处理隐式流而导致的漏报越来越受到人们的关注,例如,Egele 等人提出的对 Kirda 工作的改进方法,动态跟踪数据在 BHO 组件中的传播并发送给攻击者的过程,能够处理采用显示流和隐式流方式的数据传播[9].James 等人提出的工具 Dytan 中,全面地分析了可执行程序中的语句包括其中的隐含操作等对类型环境的影响,并且对隐式流的处理提出了一个较好的思路[10],使得 Dytan 成为一个隐式流敏感的、可通过形式化定义的通用动态数据流分析模型.由于木马间谍程序的一些特点,例如结合远程控制等技术实现以及在处理击键信息时与合法软件的相似性等,使得 Dytan 这种通用的方式不能很好地对其进行检测.本文采用静态分析方式,具有更高的代码覆盖率,同时结合数据流分析对间接跳转的目标进行计算,并提出了一种针对木马间谍程序改进的污点标记规则,对真实恶意代码和合法软件的实验证明了本文方法的有效性.

污点传播方法也被用来对其他类型的恶意代码进行检测,例如,Kruegel 等人提出的基于符号执行技术的内核级 rootkit 检测方法[22]、Stinson 等人提出的基于动态污点分析对僵尸程序(bot)的远程控制行为的检测方法[23]以及张新宇等人提出的基于本地网络的蠕虫协同检测方法等[24].与这些方法相比,本文关注击键信息窃取行为的刻画和识别.

5 结 论

使用污点传播分析方法对信息窃取软件进行检测已经成为研究热点.目前缺乏对木马间谍程序进行有效检测的方法,为此,本文提出了一种静态的隐式流敏感的木马间谍程序检测方法,能够对间接跳转语句的目标地址进行计算,并根据分支条件的操作语义给出了一个改进的污染标记规则.使用该方法对 103 个真实的恶意代码样本和 7 个合法软件进行实验,表明了该方法的有效性.需要指出的是,虽然本文只针对木马间谍程序进行了分析和验证,但本文的方法可以很容易进行扩展,以支持对采用其他信息窃取技术例如屏幕监视的恶意代码进行分析.

本文提出的木马间谍程序检测方法还可以在某些方面进行进一步的研究,主要包括将该方法与动态分析方式结合,以克服静态分析方式的一些缺点,例如:对于事件触发的非结构化流程使用动态分析得到其可能的上下文环境;结合动态分析对于静态分析无法确定的间接跳转,例如面向对象程序中的虚函数调用等进行确定以及结合动态方法对自产生(self-generating)的恶意代码进行分析等.

致谢 感谢孟涛博士、诸葛建伟博士和王铁磊博士在本文写作过程中所给予的帮助.

References:

[1] Symantec Corp. Symantec reports rise in data theft, data leakage, and target attacks leading to hackers' financial gain. 2007. http://www.symantec.com/about/news/release/article.jsp?prid=20070319_01

[2] Kapersky Lab, Nikolay G. Keylogger: How they work and how to detect them. 2007. http://www.viruslist.com/en/analysis?pubid=204791931

[3] Lavasoft. Ad-Aware. 2005. http://www.lavasoft.com/products/ad_aware_free.php

[4] Kolla PM. Spybot search & destroy. 2005. http://www.safer-networking.org/

[5] Microsoft Corp. Windows defender: Spyware protection for free. 2007. http://www.microsoft.com/athome/security/spyware/software/default.mspx

[6] Kirda E, Kruegel C, Banks G. Behavior-Based spyware detection. In: Proc. of the 15th USENIX Security Symp. Berkeley: USENIX Association, 2006. 19−19.

[7] Yin H, Song D, Egele M, Kruegel C, Kirda E. Panorama: Capturing system-wide information flow for malware detection and analysis. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007. 116−127.

[8] Denning E. A lattice model of secure information flow. Communications of the ACM, 1976,19(5):236−243. [doi: 10.1145/360051.360056]

[9] Egele M, Kruegel C, Kirda E, Yin H, Song D. Dynamic spyware analysis. In: Proc. of the 2007 Usenix Annual Conf. Berkeley: USENIX Association, 2007. 233−246.

[10] Clause J, Li WC, Orso A. Dytan: A generic dynamic taint analysis framework. In: Proc. of the 2007 Int'l Symp. on Software Testing and Analyis. New York: ACM Press, 2007. 196−206.

[11] F-Secure Inc. F-Secure malware information pages: Backdorr: W32/Hupigon. 2007. http://www.f-secure.com/v-descs/backdoor_w32_hupigon.shtml

[12] F-Secure Inc. F-Secure virus descriptions: Rbot. 2004. http://www.f-secure.com/v-descs/rbot.shtml

[13] Foster JS, Fahndrich M, Aiken A. A theory of type qualifiers. In: Proc. of the 1999 ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 1999. 192−203.

[14] Huang YW, Yu F, Hang C, Tsai CH, Lee DT, Kuo SY. Securing Web application code by static analysis and runtime protection. In: Proc. of the 13th Int'l Conf. on World Wide Web. New York: ACM Press, 2004. 40−52.

[15] Zhang Y. Research of the multi-policy security architecture based on information flow. Chinese Journal of Computers, 2006,29(8): 1453−1458 (in Chinese with English abstract).

[16] DateRecure, IDA pro disassembler. http://www.datarescue.com/idabase

[17] Wei T, Mao J, Zou W, Chen Y. Structuring 2-way branches in binary executables. In: Proc. of the 31st Annual IEEE Int'l Computer Software and Applications Conf. Washington: IEEE Computer Society, 2007. 115−118.

[18] Wei T, Mao J, Zou W, Chen Y. A new algorithm for identifying loops in decompilation. In: Proc. of the 14th Int'l Static Analysis Symp. LNCS 4634, Berlin, Heidelberg: Springer-Verlag, 2007. 170−183.

[19] Zhuge JW, Han XH, Zhou YL, Song CY, Guo JP, Zou W. HoneyBow: An automated malware collection tool based on the high-interaction honeypot principle. Journal on Communications, 2007,28(12):8−13 (in Chinese with English abstract).

[20] Microsoft Corp. Malicious software encyclopedia: Win32/IRCBot. 2007. http://www.microsoft.com/security/encyclopedia/details.aspx?name=Win32%2fIRCBot

[21] Symantec Corp. Spyware KeyLogger. 2007. http://www.symantec.com/security_response/writeup.jsp?docid=2004-033116-4256-99

[22] Kruegel C, Robertson W, Vigna G. Detecting kernel-level rootkits through binary analysis. In: Proc. of the 20th Annual Computer Security Applications Conf. Washington: IEEE Computer Society, 2004. 91−100.

[23] Stinson E, Mitchell JC. Characterizing bots' remote control behavior. In: Proc. of the 4th GI Int'l Conf. on Detection of Intrusions & Malware and Vulnerability Assessment. LNCS 4579, Berlin, Heidelberg: Springer-Verlag, 2007. 89−108.

[24] Zhang XY, Qing SH, Li Q, Li DZ, He ZH. A coordinated worm detection method based on local nets. Journal of Software, 2007, 18(2):412−421 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/412.htm [doi: 10.1360/jos180412]

附中文参考文献:

[15] 张阳.基于信息流的多安全策略操作系统架构研究.计算机学报,2006,29(8):1453−1458.

[19] 诸葛建伟,韩心慧,周勇林,宋程昱,郭晋鹏,邹维.HoneyBow:一个基于高交互式蜜罐技术的恶意代码自动捕获器.通信学报,2007,28(12):8−13.

[24] 张新宇,卿斯汉,李琦,李大治,何朝辉.一种基于本地网络的蠕虫协同检测方法.软件学报,2007,18(2):412−421. http://www.jos.org.cn/1000-9825/18/412.htm [doi: 10.1360/jos180412]

李佳静(1979－),女,黑龙江大庆人,博士,讲师,主要研究领域为恶意代码分析,软件脆弱性分析.

梁知音(1979－),女,助理研究员,主要研究领域为恶意代码分析.

韦韬(1975－),男,博士,助理研究员,主要研究领域为反编译,软件脆弱性分析,恶意代码分析.

邹维(1964－),男,研究员,博士生导师,主要研究领域为网络与信息安全.

毛剑(1978－),女,博士,主要研究领域为网络与信息安全.