# 基于污点分析的源代码脆弱性检测技术

孔德光[1],郑 烇[1],帅建梅[1],陈 超[1],葛 瑶[2]

[1](中国科学技术大学 自动化系,安徽 合肥 230027)
[2](中国科学院 合肥物质科学研究院 网络中心,安徽 合肥 230031)
E-mail:kdg@mail.ustc.edu.cn.

**摘 要**:基于源代码的静态分析技术是检测软件脆弱性的一种重要手段.针对不可信数据输入导致软件脆弱性的问题,提出一种基于污点分析的脆弱性检测方法.通过跟踪程序参数、环境变量等各种外部输入,标记输入的类型,在构造控制流图基础上,利用数据流分析中的相关信息,污点传播至各类脆弱性函数,从而解决缓冲区溢出、格式化字符串等问题.利用控制流、数据流分析的相关信息,提高了准确率,降低了漏报率.实验表明,该技术是一种有效的脆弱性分析方法.

**关 键 词**:污点分析;控制流;数据流;脆弱性
**中图分类号**:TP393.08 **文献标识码**:A **文 章 编 号**:1000-1220(2009)01-0078-05

## Source Code Vulnerability Detection Technology Based on Taint Analysis

KONG De-guang[1], ZHENG Quan[1], SHUAI Jian-mei[1], CHEN Chao[1], GE Yao[2]
[1](*Department of Automation, University of Science & Technology of China, Hefei* 230027, *China*)
[2](*Network Center, Hefei Institute of Physical Science, Chinese Academy of Sciences, Hefei* 230031, *China*)

**Abstract**: Static analysis technology is a significant method to detect software vulnerabilities. To cope with the problem of untrusting data inputs leading to software vulnerabilities, presents a vulnerability detection method based on taint analysis. It tracks various kinds of input including program parameters and environment variables , marks the type of input, after constructing the control flow graph, makes use of dataflow information, propagating the taint data to the vulnerability functions, to settle the problem of buffer overflow and format string. It utilizes the related information of control flow and dataflow during this process, thus improves the accuracy and decreases the false negatives. It is proved by experiment that this technology is an effective vulnerability analysis method.

**Key words**: taint analysis; control flow; dataflow; vulnerability

## 1 引 言

随着信息社会对软件依赖程度日益提高,提高软件可靠性、减少软件脆弱性变得愈加重要.国家计算机网络应急技术处理协调中心(CNCERT/CC)2006 年网络安全工作报告中指出 USCERT/CC 组织 2006 年全年收到信息系统安全漏洞报告数 8064 个,与 2005 年同期相比增加 2074 个.黑客就是利用缓冲区溢出、数组越界等软件中的脆弱性,来达到破坏系统和获得未经授权的访问控制的目的.2007 年上半年艾妮病毒大面积传播就是利用了 Windows 系统漏洞 MS07-017 中动态光标远程执行代码漏洞(ANI 漏洞).源代码脆弱性分析技术通过对源代码进行扫描分析检测出程序中潜在的脆弱性,在漏洞挖掘中扮演着重要的角色.

已有的静态分析工具,按照其分析机理的不同,有基于词法分析工具,如 Its4[1],基于语法分析的工具,如 Boon[2],基于模型检测的工具如 Mops[3]等.由于程序的不可判定性和分析时近似计算,这些工具都会产生一定的误报和漏报.与上述技术相比,污点分析技术通过跟踪污点数据的输入,记录数据的流向,从而分析得到违反规则或者相应约束的软件脆弱性代码,针对性更强,相对更加高效准确.

动态污点分析的解决方法,通过在程序执行过程中跟踪变量、存储单元、寄存器的值,达到跟踪攻击路径,获取漏洞信息的目的[4-5].静态污点分析是将污点传播过程嵌入到类型分析中,将污点视为扩展的类型信息,通过编译器的类型分析,获得丰富的类型信息,从而在类型推断(type-inference)和类型限定(type-qualification)中完成污点数据传播的功能,达到分析软件脆弱性的目的.例如,Berkerly 大学 2002 年推出的用于检测 C 语言程序脆弱性的工具 Cqual[6]和 2006 年推出的 Cqual 工具的扩展 Oink[8],是基于类型分析和限定的思想.这种方法利用编译器类型信息,分析较复杂,需要较强的类型分析和推断算法,并且很多情况下是流不敏感的(flow-insensitive)[9].

收稿日期:2007-08-23 基金项目:国家"八六三"高技术研究发展计划基金项目(2006AA01Z449)资助. 作者简介:**孔德光**,男,1983 年生,博士研究生,研究方向为信息安全、软件安全;**郑 烇**,男,1970 年生,副教授,研究方向为网络安全,计算机网络;**帅建梅**,女,1961 年生,高级工程师,研究方向为信息安全;**陈 超**,男,1983 年生,硕士研究生,研究方向为网络安全;**葛 瑶**,女,1984 年生,硕士研究生,研究方向为 Web 应用与软件安全.

本文利用污点类型信息,在分析程序控制流图的基础上,融合程序的数据流信息(包括到达一定值,左值,右值),完成相关污点标记和判定过程.充分利用了程序的控制流、数据流信息,替代了繁琐的类型推导和限定的过程,并可以支持流敏感的分析.特点在于利用了控制流、数据流分析的相关技术,提取程序关键信息,有效提高污点分析的精度.经实验表明,这种方法漏报率较低,能够检测到大多数的程序脆弱性,还可以为语义分析提供脆弱性路径签名,不失为一种有效的方法.

## 2　污点传播的基本思想和原理

### 2.1　污点的来源

污点源指可能引起程序产生各类安全问题的数据来源.用污点 tainted 标记这样的数据项.它包括:键盘的输入,读写磁盘(文件)的输入,网络接口的输入,客户端 Web 参数等等.通过构造不可信任用户的输入,触发程序的漏洞,达到攻击利用程序的目的.因此,跟踪程序的输入是非常必要的.

### 2.2　污点传播的基本思想

通过对不信任的输入数据做标记,静态跟踪程序运行过程中污点数据的传播路径,检测使用污点数据的不安全方式,用这种方法可以检测到敏感数据(如字符串参数)被改写而造成的缓冲区溢出、格式化字符串等问题.当检测到一个攻击时,污点分析技术可以提供详细的攻击过程,给出由于污点数据所导致的漏洞被利用的过程.污点及其传播路径信息还可以用来产生脆弱性签名.

**定义 1.** 代数结构格 L 由值集合和两个运算组成(交运算 $\cap$,并运算 $\cup$).L 满足下列性质:

(1) 对于所有的 x,y∈L,存在唯一的 z∈L,w∈L,使得 x $\cap$ y=z 且 x $\cup$ y=w(封闭性);

(2) 对于所有的 x,y∈L,x $\cap$ y=y $\cap$ x 并且 x $\cup$ y=y $\cup$ x(交换性);

(3) 对于所有的 x,y,z∈L,(x $\cap$ y) $\cap$ z=x $\cap$ (y $\cap$ z)并且(x $\cup$ y) $\cup$ z=x $\cup$ (y $\cup$ z)(结合性);

(4) L 中存在两个唯一的底元素(bottom)$\perp$,顶元素(top)T,使得所有 x∈L,x $\cap$ $\perp$ = $\perp$ 并且 x $\cup$ T=T(存在唯一的底元素和顶元素).

代数结构格的元素可以是变量、表达式或者一个过程可能执行的其他程序设计结构的抽象性质.

**定义 2.** 在格上定义了偏序关系,x≤y,当且仅当 x $\cap$ y=x.该偏序关系满足下列性质:

(1) 对所有的 x,y,z,如果 x≤y,y≤z,则 x≤z(传递性);

(2) 对所有的 x,y,如果 x≤y,y≤x,则 x=y(对称性);

(3) 对所有的 x,有 x≤x(自反性).

对于有污点(tainted)和无污点(untainted)两种数据类型而言,可以认为是满足偏序关系的格上的两个元素,满足 untainted≤tainted,即 untainted 类型作为 tainted 类型的子类型.当数据源要求污点数据(tainted)时,可以将没有污点的数据作为输入使用.而数据源要求非污点数据(Untainted)时,不可以将有污点的数据作为输入使用.更一般地,包含类型限定符 Q1 和 Q2 的格为 Q1≤Q2,那么对于任意类型 τ,$Q_1\tau$ 和 $Q_2\tau$ 是两个新的限定符类型,并且 $Q_1\tau$ 是 $Q_2\tau$ 的子类型(记作 $Q_1\tau \leqslant Q_2\tau$).故一个 $Q_1\tau$ 类型的变量能够当成一个 $Q_2\tau$ 类型的变量来使用,反之不然.

**定理 1.** 令(P,≤)是任意的有限格.令($2^N$,⊆)是由 N 的有限子集形成的满足包含关系的格.那么(P,≤)可以嵌入到($2^N$,⊆)中,即存在映射,$\varphi: P \to 2^N$,使得 a≤b⇔$\varphi(a) \subseteq \varphi(b)$对于任意 x∈P,$\varphi(x)$是 N 的有限子集.

定理证明[10]略.对于形如函数调用语句 tgt_1_2=call(src_1,src_2,…); src_1 对应参数 arg1,src_2 对应参数 arg2,tgt_1_2 为返回值,call 表示函数调用,可得函数参数与返回值之间污点传播关系.若 src_1,src_2,tgt_1_2∈P,构造 $\varphi$,取"_"标记的数值部分,可得 $\varphi$(src_1)={1}, $\varphi$(src_2)={2},$\varphi$(tgt_1_2)={1,2};那么由{1}⊆{1,2},{2}⊆{1,2}可得,src_1≤tgt_1_2, src_2≤tgt_1_2.若不能构造满足包含关系的子集,那么函数参数到返回值未进行污点传播,其语义不反映污点作用.

### 2.3　脆弱性签名

使用四元组序列$(P,T,x,c)$表示漏洞位置,其中 P 代表源程序,x 是输入数据,c 是脆弱性条件,T 是 x 在 P 上的执行路径.那么漏洞签名可以使用 $T(P,c)$来表示,即由输入 x 所获得的程序 P 上的执行路径为 T.脆弱性条件 c 在执行路径 T 上被检查.如果 T 满足脆弱性条件,记作$|T|=c$.定义脆弱性语言 $L_{p,c}$,描述对于程序 P,输入 x,产生执行路径满足脆弱性条件 c.令 $\sum^*$ 表示程序 P 的输入域.$L_{p,c} = \{x \in \sum^* | T(P,c) = c\}$.那么一个漏洞签名满足下列属性:

$$match(x) = \begin{cases} \text{exploit} & \text{when } x \in L_{p,c} \\ \text{benign} & \text{when } x \notin L_{p,c} \end{cases}$$

其中 exploit 代表满足脆弱性条件,产生漏洞.benign 代表不满足脆弱性条件,不可以被利用.

## 3　污点分析系统(TAS-Taint Analysis System)构架设计

根据上述基本原理,给出污点传播分析系统的构架设计,如图 1 所示.

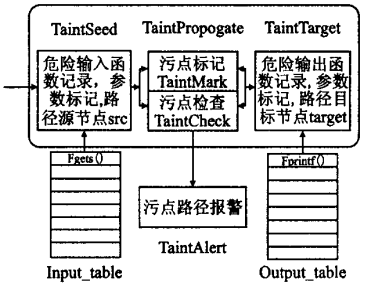


图 1　系统构架图

Fig. 1　Architecture of system

(1) 污点来源模块为 TaintSeed:记录危险输入函数并标记其参数.将危险输入函数作为路径跟踪的源节点(src).函

数族的参数是污点传播关注的焦点.人工识别外界输入的参数,通常将指向缓冲区数据的指针标记为 tainted 类型.例如,函数原型 int fread(void tainted * ptr, int size, int nitems, FILE *stream)中,将缓冲区指针 ptr 标记为 tainted.对于输入函数的返回值,若返回值指示该操作是否成功,返回成功或者失败标志或者其它整形常量,返回值可以忽略,不受污点的感染.Input_table 中记录污点来源的快照,包括:污点来源位置(文件名);函数名,参数,返回类型,如表 1 所示.

表 1 污点输入函数列表(部分)

Table 1 Input function for tainted data(partly)

| |
|---|
| int read(int fildes, void tainted * buf, int nbyte); |
| int recv(int s, void tainted * buf, int len, int flags); |
| int fread (void tainted * ptr, int size, int nitems, FILE * stream);… |

(2) 污点影响目标节点模块 TaintTarget:记录危险输出函数,参数标记.将这些危险输出函数作为路径跟踪的目标节点(target).Output_table 记录导致程序出现漏洞的脆弱性函数.以格式化字符串 prinf 系列库函数为例,验证其参数是否被污染,相当盂 hook 了 printf 函数.验证系统调用的参数和其它函数也是同样的原理,如表 2 所示.

表 2 输出脆弱性函数列表(部分)

Table 2 Output function for tainted data (partly)

| |
|---|
| int fprintf(FILE *stream, const char *format, …); |
| int printf(const char *format, …) ; |
| int sprintf(char *s, const char *format, …); |

(3) 污点传播过程 TaintPropagate:包括污点标记 TaintMark 和污点检查 TaintCheck 两个模块,不能通过 TaintCheck 时,会引起污点路径报警 TaintAlert.

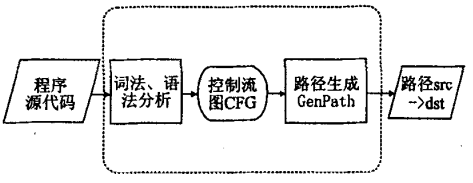


图 2 污点路径生成图

Fig. 2 Graph for tainted path system

上述过程中源节点(src)到目标节点(tgt)的数据流向,依赖于对程序结构的分析,在得到控制流图的基础上,调用路径生成模块(GenPath),获得从污点来源到目标节点的路径,如图 2 所示.

## 4 污点分析系统(TAS-Taint Analysis System)核心算法

### 4.1 基于控制流图(C FG)的路径生成模块

源节点 src 到目标节点 tgt 路径生成 GenPath 算法如图 3 所示.设有 n 个输入源节点,m 个目标脆弱性函数节点,那么(m * n)对源节点和目标节点之间路径可以通过遍历控制流图的基本块得到.具体寻找路径时采用图可达性分析(graph-reach analysis)算法,使用广度优先搜索策略遍历 CFG,生成从污点源节点到污点目标(即脆弱性函数)的所有基本路径.

CFG 中每个子函数对应一个控制流图,使用由(源文件名,子函数名,参数列表,返回值)四元组构成的函数签名进行标识.每个控制流图由若干个基本块组成.对于源节点和目标节点不在同一个函数的情况,函数展开采用按需展开(on-demand)的启发式算法.当污点数据作为参数(或者全局变量),在被调用函数中出现并发生改变(即产生定值-引用相关操作)时,该函数调用模块的控制流图 CFG 才具体展开;否则,不展开.

**输入**:源节点 src:输入污点函数;目标节点 tgt:脆弱性函数

**输出**:src 到 tgt 的路径 path[path_num][step_num],path_num 记录路径编号,step_num 记录每条路径步数

**算法**

```
1 block_src=find_block(src), block_dst=find_block(tgt),
                          //识别源节点和目标节点所在基本块
2 if(block_src == block_tgt)          //在同一个基本块内
3   Path[][]=block_traverse(block_src,src,tgt)
                          //直接在基本块内寻找从 src 到 tgt 的路径
4 else if(Equal_fun(block_src,block_tgt ) //同一个函数内
5   path[][]=func_traverse(func_name,src,tgt);
                          //寻找该函数内块之间的路径
6 else                                //不在同一个函数体内
7   path[][]=fun_inter_traverse(fun_src,fun_dst,src,
                                                  tgt)
          //按需展开函数调用,找出不同函数调用之间的路径
```

图 3 路径产生算法流程

Fig. 3 Algorithm for path generation

### 4.2 污点传播模块

#### 4.2.1 污点传播 TaintPropogate

跟踪污点数据的传播过程,用于确定其它数据是否被污染,主要跟踪有关污点的数据操作类指令.污点正向传播过程是在控制流图中跟踪 tainted 和 untainted 数据的变化过程.污点传播分为污点标记 TaintMark 和污点检查 TaintCheck 两个子过程.在污点分析过程中,采用了流敏感(flow-sensitive)的分析技术.src 和 tgt 分别表示源和目的数据源.taint(stmt,tgt)、taint(stmt,src)={tainted,untainted}分别表示 stmt 语句中 src、tgt 数据是否被污染.taint(stmt,tgt)=taint(stmt,src)表示 stmt 语句中 tgt 数据的污点类型由 src 数据污点类型决定.若 src 为 tainted,则 tgt 为 tainted;src 为 untainted,则 tgt 为 untainted.每条有关数据操作的语句(如赋值、函数调用等)中至少有一个源数据 src,通常为右值(Right-hand value);一个目标数据 tgt,通常为左值,(left-hand value);反映数据流分析中数据的流向.有多个源数据的情况也可以归约为多个一对一的从源数据到目标数据的数据

流向的情况.

4.2.2 数据流驱动的污点标记过程 TaintMark

**输入**:当前语句 stmt

**输出**:标记该条语句中从 src 到 tgt 的污点传播过程

**算法**:

```
1 stmt_type=get_type(stmt) //获得当前语句的类型
2 src=get_src(stmt); tgt=get_tgt(stmt);//获得源数据,
                                        目标数据
3 switch(stmt_type):
4   case "reach-definition" : //赋值语句"target=source;"
或简单线性运算 target=f(source), 例如形如 target=source
        +3 的指针偏移等,f 表示 tgt 是 src 的线形运算.
5   taint(stmt,tgt)=taint(stmt,src), break;
6   case "call_func" ://函数调用 tgt=call (src); src 为参
数 arg,tgt 为返回值,call 表示函数调用,使用定理 2.1 的方
                                        法判定
7   if (tgt 反映 src 作用是否成功) taint(stmt,tgt)=untaint-
ed. //tgt 反映 src 作用是否成功,表示函数返回值可能为 0,
        1,或者字符串成功操作的长度,而与 src 类型无关.
8 else if (tgt 由 src 变化而来) taint(stmt,tgt)=taint(stmt,
                                        src);
9   break;
10   case "argu0_1": taint(stmt,tgt)=taint(stmt,src) ;
                                        break;
//同一个函数相关的参数(arg_0,arg_1 等等),例如 strcpy
(tgt,src). 不相关的参数不满足此条件,例如 strcat
                                        (tgt,src);
11 default: break; //缺少 src 或者 tgt,不进行处理
```

图 4　污点标记算法流程

Fig. 4　Algorithm for taintmark

污点标记算法 TaintMark(stmt)算法如图 4 所示. taint(stmt,src)用于分析 stmt 语句中 src 的污点类型. taint(stmt,tgt)的确定,原理同 taint(stmt,src)的确定方法,只需将 tgt 换成 src 即可. 该过程中用到了数据流分析中的合流运算的思想,定义见 4.2.3.

4.2.3 数据流分析中的合流运算 Meet over all paths (MOP)

用于确定某点数据是否被污染,其中令 Tgt=f1(src_1), tgt=f2(src_2),…,tgt=fi(source_i),fi 表示对 source_i 进行作用,可以是赋值、算术加减、字符串连接或者函数调用等运算. 采取保守分析的方式,定义:

$$\text{taint(stmtgt)} = \text{meet(f1,f2,}\cdots\text{,fi)} = \begin{cases} tainted, & if\ \exists\ \text{taint(stmt},fi(src\_i)) = tainted, \\ & i = 1,2,\cdots,n \\ untaited, & if\ \forall\ \text{taint(stmt},fi(src\_i)) = untained, \\ & i = 1,2,\cdots,n \end{cases}$$

4.2.4 数据流传播中的污点检查过程 TaintCheck

污点检查算法 TaintCheck(stmt)如图 5 所示. 其过程与污点标记算法(TaintMark)相似,"case"条件相同. 例如 case "reach-definition"对应赋值或简单线性运算,要求 taint(src)≤taint(tgt)满足偏序关系. 由于到达目标结点定值点的路径可能多条,单独每条语句满足规则为 taint(src)≤taint(tgt),在多次定值并且合流后不一定满足偏序关系. 若不满足,则出现污点报警(TaintAlert).

**输入**:当前语句 stmt

**输出**:检查该条语句从 source 到 target 是否满足偏序关系. 满足,正常退出;不满足,给出污点路径.

**算法**:

```
1 stmt_type=get_type(stmt) //获得当前语句的类型
2 src=get_src(stmt); tgt=get_tgt(stmt); //获得源数据,
                                        目的数据
3 switch(stmt_type):
4case "reach-definition" :
5   check(tgt,src,stmt) break;
6 case "call_func" :
7   if (tgt 由 src 变化而来) check(tgt,src,stmt);
8   break;
9 case "argu0_1": check(tgt,src,stmt); break;
10 default: break;
```

图 5　污点检查算法流程

Fig. 5　Algorithm for taint check

## 5　路径传播算法的运行示例程序解析

给出一个示例程序,图 6 所示,按照污点路径传播算法进行运行分析. 污点输入函数 char * getenv(tainted const char * name);污点参数来源于环境变量"User",该函数返回值记录了有污点参数 name 输入后的结果,故而返回值为 tainted.

```
1 void main(void)
2 {char *s=new char[18]; char * t=new char[18];
3   if((s = getenv("USER"))) //获取环境变量的取值
4 t=s;
5   else
6     strcpy(t,"untainted");
7   printf(t);}
```

图 6　示例程序

Fig. 6　An example of program

污点输出函数有 strcpy(char * dst,char * src)和 printf(untainted const char * fmt, …);其中 7 行处参数字符串 t 要求是 untainted. strcpy 中由于 src 是常数字符串,故而 t 是无污点的. 在污点标记(TaintMark)中,4 行 t 为 taintd,6 行 t 为 untainted,故两个分支到达 7 行合流后为 taintd. 污点判定过程(TaintCheck)中,发现 7 行 tainted≤untaintd,不满足偏序关系,给出污点报警,并给出相应的路径 1-2-3-4-5-7.

## 6　实验结果

本文选取 3 个软件包来验证该污点传播方法的有效性. 这 3 个软件包在开源社区中广泛使用,作为常见的网络应用层工具的源代码,具有一定的代表性. 测试的实验环境:Pentium(r) 4, 3.0G CPU, 256M 内存,Linux(Red Hat9.0),

GCC3.2.1.表3给出了对不同源代码包进行测试后的结果.

```
while (fgets(buf,sizeof buf, f)){ lreply(200,buf);…}
void lreply(int n,char * fmt,…)
{…vsnprintf(buf, sizeof buf, fmt, ap);…}
```

图 7 wu-ftpd 2.6.0 中的一段脆弱性代码

Fig. 7 A section of vulnerability code in wu-ftpd 2.6.0

包括应用程序名,版本号,描述信息,总行数(LOC),静态检测需要的时间(Time),工具报出的总漏洞数(LOE),检查后发现的实际漏洞数目(RBug).例如通过该方法,发现的 wu-ftpd2.6.0 软件包中一个格式化字符串问题如图 7 所示.图 7 程序中的"lreply(200,buf);"应该被语句"lreply(200,"%s", buf)"所替代,编程者省略了"%s",从而会引起安全漏洞.

表 3 不同软件包的测试结果

Table 3 Test result for different software package

| Program | Ver. | LOC | Time | LOE | RBug |
|---|---|---|---|---|---|
| wu-ftpd | 2.6.0 | 13k | 12s | 64 | 5 |
| muh | 2.05d | 3k | 5s | 12 | 1 |
| apache | 1.3.12 | 33k | 43s | 0 | 0 |

实验结果分析:从上述实验数据可以看出,基于控制流、数据流分析的污点传播算法可以有效地用于发现源代码中的脆弱性.尽管发现的漏洞报告中包含着误报,报告的总条数还是较少的,从这些较少的报告数中找出真正的漏洞,已经大大减少了工作量.由于分析时采用了保守分析的方法,加之静态分析本身不可能获取所有运行时的数据,不可能达到对数据完全定值的效果,完全精确是不可能的.漏报率和误报率是一对矛盾,加大分析深度,尽可能多报错误,可以降低漏报率,但同时误报率会提高.而增加约束条件,少报错误,显然会使误报率降低,但同时漏报率不可避免要提高.该方法在漏报率和误报率之间做了折衷,兼顾效率,取得较好的结果.该方法存在少量漏报,例如:if(x==0) y=0; if (x==1) y=1; 和 y=x;有相同的语义,如果 x 为污染数据,使用前面的 if 语句,不用标记 y 为被污染的,存在漏报.本来是不信任的输入域被标记为信任的输入域会导致漏报,或者是程序读取从不信任区域转化而来的信任数据也存在漏报.存在误报,由于使用保守的分析方法,将合流 MOP 后的污点类型记录下来作为后来路径传播的源头,而恰有些路径是没有污点的,这时要通过回溯合流前没有污染的路径,以便区分污染和非污染路径.

## 7 总结与展望

本文给出了一种基于污点分析的源代码脆弱性检测方法,通过跟踪污点的数据输入,用于发现诸如格式化字符串、缓冲区溢出程序漏洞.通过选定关键输入和输出函数来缩小检测范围,利用函数名、变量名匹配并结合控制流图中的路径搜索寻找代码中的漏洞,并给出脆弱性路径报告可用于脆弱性签名.实验表明,漏报率较低,性能较好.下一步将扩展偏序关系,使其支持更多种污点分析的检测和约束关系,改进实现更有效的污点传播算法.

**References:**


[1] John Viega, Bloch J T. ITS4:a static vulnerability scanner for C and C++ code[C]: Annual Computer Security Applications Conference, December 2000.

[2] Wagner D, Foster J, Brewer E, et al. A first step towards automated detection of buffer overrun vulnerabilities[C]. Network and Distributed Systems Security Conference, San Diego, CA, February 2000.

[3] Chen H, Wagner D. MOPS: an infrastructure for examining security properties of software[C]. Proc. 9th ACM Conf. Computer and Communications Security (CCS2002), ACM Press, 2002,235-244.

[4] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]. 12th Annual Network and Distributed System Security Symposium, Feb. 2005.

[5] Cheng W, Zhao Q, Yu B, et al. Taint trace: efficient flow tracing with dynamic binary rewriting[C]. Proocedings of the 11th IEEE Symposium on Computers and Communications, June, 2006.

[6] Cqual[EB/OL]. http://www.cs.umd.edu/~jfoster/cqual/, August 2007.

[7] Zhang X, Edwards A, Jaeger T. Using CQUAL for static analysis of authorization hook placement[C]. Proceedings of the Eleventh Usenix Security Symposium, August 2002.

[8] Oink [EB/OL]. http://www.cubewano.org/oink, August 2007.

[9] Jeffrey S Foster. Type qualifiers: lightweight specifications to improve software quality[D]. University of California, Berkeley, December 2002.

[10] Umesh Shankar, Kunal Talwar, Jeffrey S Foster, et al. Detecting format-string vulnerabilities with type qualifiers[C]. 10th USENIX Security Symposium, August 2001.

[11] Steven S Muchnick. Advanced compiler design and implementation[M]. Morgan Kaufmann Publishers, Fransisco, USA, 1997,10-65.

# 基于污点分析的源代码脆弱性检测技术



作者： 孔德光， 郑烃， 帅建梅， 陈超， 葛瑶， KONG De-guang， ZHENG Quan， SHUAI Jian-mei， CHEN Chao， GE Yao

作者单位： 孔德光,郑烃,帅建梅,陈超,KONG De-guang,ZHENG Quan,SHUAI Jian-mei,CHEN Chao(中国科学技术大学,自动化系,安徽,合肥,230027)， 葛瑶,GE Yao(中国科学院,合肥物质科学研究院,网络中心,安徽,合肥,230031)

刊名： 小型微型计算机系统 ISTIC PKU

英文刊名： JOURNAL OF CHINESE COMPUTER SYSTEMS

年，卷(期)： 2008,30(1)

被引用次数： 2次

## 参考文献(11条)

1. John Viega;Bloch J T ITS4:a static vulnerability scanner for C and C++ code 2000

2. Wagner D;Foster J;Brewer E A first step towards automated detection of buffer overrun vulnerabilities 2000

3. Chen H;Wagner D MOPS:an infrastructure for examining security properties of software 2002

4. Newsome J;Song D Dynamic taint analysis for automatic detection,analysis,and signature generation of exploits on commodity software 2005

5. Cheng W;Zhao Q;Yu B Taint trace:efficient flow tracing with dynamic binary rewriting 2006

6. Cqual 2007

7. Zhang X;Edwards A;Jaeger T Using CQUAL for static analysis of authorization hook placement 2002

8. Oink 2007

9. Jeffrey S Foster Type qualifiers:lightweight specifications to improve software quality 2002

10. Umesh Shankar;Kunal Talwar;Jeffrey S Foster Detecting format-string vulnerabilities with type qualifiers 2001

11. Steven S Muchnick Advanced compiler design and implementation 1997

## 本文读者也读过(4条)



## 引证文献(2条)