一种针对非控制数据攻击的改进防御方法*

刘小龙[a,b], 郑 滔[a,b]
(南京大学 a. 计算机软件新技术国家重点实验室; b. 软件学院, 南京 210093)

摘 要: 非控制数据攻击(non-control-data attack)是一种有别于传统攻击模式的新方式,通过窜改系统中的安全关键数据,实现不改变程序控制流程进行攻击。针对已有的防御措施中静态分析方法依赖源代码,而动态分析方法存在严重的误报漏报,提出了一种指针污点分析方法。该方法基于动态污点分析技术,标记内存数据的污染属性、指针属性,跟踪标记信息在程序执行时的传播,监控是否存在指针的非法解引用(dereference)。设计实现了原型系统 DPTA,通过实验证明本方法可以有效地防御控制数据攻击和大部分非控制数据攻击,并减少误报漏报。

关键词: 非控制数据攻击; 程序安全; 动态污点分析; 指针着色; 内存破坏
中图分类号: TP309 **文献标志码**: A **文章编号**: 1001-3695(2013)12-3762-05
doi:10.3969/j.issn.1001-3695.2013.12.063

Improved defense method against non-control-data attacks

LIU Xiao-long[a,b], ZHENG Tao[a,b]
(a. *State Key Laboratory for Novel Software Technology*, b. *Software Institute*, *Nanjing University*, *Nanjing* 210093, *China*)

Abstract: Non-control-data attack is a new attack method which corrupts security data instead of the target program's control data. Existing static analysis methods require recompiling. Dynamic analysis methods have high false-positive and false-negative rate. This paper proposed pointer taint analysis method based on dynamic taint analysis. This method used taint tag and pointer tag to mark memory data. It propagated these two tags during program's execution, and detected attacks when an invalid pointer was dereferenced and this pointer has been manipulated by attackers. It implemented a tool based on dynamic binary instrumentation framework Pin. The results of the experiment show this method can detect control-data attacks and most of non-control-data attacks.

Key words: non-control-data attack; programming security; dynamic taint analysis; pointer taint; memory corruption

0 引言

随着计算机和网络技术的飞速发展,各类计算机病毒、蠕虫、恶意程序对成千上万的计算机系统产生了巨大威胁,程序安全问题已经成为信息化时代人们所面临的严峻问题之一。著名的 Morris、CodeRed、Slammer 等蠕虫利用系统的漏洞在互联网进行了快速传播,给全球的电脑用户带来了巨大的经济损失。计算机系统的漏洞使得攻击者可以进行大量的内存破坏攻击。常见的攻击方式包括缓冲区溢出攻击、堆溢出攻击、格式化串攻击、ROP 攻击[1]等。

内存破坏攻击一般利用程序漏洞,覆写某处内存位置来达到改变控制流的目的,最终程序将会执行攻击者设定好的一段攻击代码。经常被覆写的程序控制数据包括返回地址、函数指针等。这类攻击被称为控制数据攻击(control-data attack),因为攻击目标是控制程序执行流程的数据。这种攻击是最常见的,目前已经有一系列比较成熟的防御方式,如哨兵位检查、栈保护、地址随机化等。

近期的研究表明,有一类攻击通过覆写非控制数据,如用户身份信息、配置信息等安全数据,同样可以达到攻击的目的,这类攻击称为非控制数据攻击[2,3]。实施这种攻击需要对程序的数据区域非常熟悉,并据此构造出巧妙的攻击数据,虽然攻击难度比控制数据攻击大很多,但是由于没有修改控制数据,现有的防御措施无法检测出,所以其对系统安全仍然是一个巨大的威胁。

动态污点分析(DTA)[4~7]是近些年得到广泛关注的一种攻击检测技术,其主要思想是将外来的数据标记为受污染的(taint),跟踪这类数据在程序中的运行,并把与外来数据具有传播关系的数据同样标记为污点,最后在安全敏感处(如跳转指令)进行检查。本文基于这一方法,提出了一种改进型方法——指针污点分析,可以用来防御非控制数据攻击。这一方法对于每一块内存数据,使用污点位、指针位两个标志位来标志。程序运行时同时传播这两类信息,当外部数据被用做指针,并且没有与合法指针结合,进行解引用(dereference)的时候,即认为是一次攻击。原型系统是基于动态二进制插桩工具 Pin[8],实验结果表明其可以防御控制数据攻击和大部分非控制数据攻击,并且可以有效地减少误报和漏报。

1 背景知识

1.1 控制数据攻击与非控制数据攻击

控制数据攻击通常利用缓冲区溢出等系统漏洞来修改内

收稿日期: 2013-04-08; **修回日期**: 2013-05-26 **基金项目**: 国家自然科学基金资助项目(61073027,61105069)

作者简介: 刘小龙(1988-),男,江苏南通人,硕士,主要研究方向为程序安全 (mg1032008@software.nju.edu.cn); 郑滔(1966-),男,教授,主要研究方向为程序安全.

存区域的数值,如返回地址、函数指针等,这一数值将会被载入程序计数器进行下一步执行。代码 1 中服务器端读取一个网络请求,并调用相应的处理函数。攻击者通过输入大于 reqbuf 最大字节数的数据,从而覆写处理函数指针,最终可以调用攻击者指定的攻击函数。

非控制数据攻击也会利用系统类似的漏洞来进行攻击,不过它会通过修改系统的关键数据来达到目的。代码 2 中,变量 name 存储的是服务器端名称字符串的地址,通过输入大于 cl_name 最大字节数的数据,攻击者可以修改 name 数值,因此来自服务器端的回复字符串中将会包含客户端信息和攻击者指定的一块内存区域的信息,这将会造成服务器的信息泄露,攻击者根据获取的信息可以进行下一步攻击。

代码 1　控制数据攻击

```
struct req {
   char reqbuf[64];
   void( *handler)(char *);
};
void do_req(int fd,
   struct req *r)
{
   //now the overflow
   read(fd, r-reqbuf, 128);
   r→handler(r→reqbuf);
}
```

代码 2　非控制数据攻击

```
void serve(int fd){
   char * name = MyHost;
   char cl_name[64];
   char svr_reply[1024];
   //now the overflow
   read(fd, cl_name, 128);
   sprint(svr_reply, "hello %s,
      I am %s", cl_name, name);
   svr_send(fd, svr_reply, 1024);
};
```

Chen 等人[2]是第一个提出非控制数据攻击会像控制数据攻击一样具有很大的危害性,指出可以通过覆写很多不同类型的关键数据来达到攻击的目的。这些数据包括配置数据、用户身份数据、用户输入信息和决策数据等。实验表明 Wu-ftpd、Null httpd 等一系列应用程序都会受到此类攻击影响。

1.2　相关防御方法

针对非控制数据攻击,目前存在一些防御方式[9~14]。其中大部分都是静态方法,需要程序源代码进行重新编译,无法应用于商业软件。也有一些动态分析解决方案,但是存在较严重的误报漏报。

文献[9]提出了 C 语言的一个扩展语义——YARRA,通过将程序中的关键数据声明为一种特殊类型,只有类型匹配的指针才能访问这类数据;实现了一个编译器的原型,编译后的程序可以有效地防御非控制数据攻击。

ValueGuard[10]在程序数据前添加哨兵位,将原始数据和哨兵位数据包装为结构体,每次运行时检查该哨兵位。这种方式同样需要对源程序进行重新编译。

数据空间随机化(data space randomization, DSR)[11]将内存中数据的存储内容随机化,针对不同的变量使用不同的掩码,与实际数值进行异或操作,当读取的时候再使用该掩码异或操作获取实际数值。攻击者通过覆写关键数据进行攻击时,由于不同变量掩码不同,因此实际写入的将会是垃圾数据,无法实施攻击。

文献[12]针对非控制数据攻击多数都存在非法指针解引用的现象,设计了一种基于边界检查的防御方法,但是文献[3,15]指出这存在很严重的误报漏报。

文献[13]通过在硬件层面给寄存器、内存添加标记位,这样可以高效地进行污点标记,不过这需要全新的硬件支持,实用性不强。

2　防御原理

基础的动态污点分析(DTA)针对控制数据攻击非常有效,但是无法防御非控制数据攻击。研究发现,控制数据攻击和大部分非控制数据攻击都具备相同的特征,向攻击者构造的地址写入数据,或者读取攻击者构造的地址数据。概括地说,大部分攻击都依赖于一个不安全的指针解引用[12,13],而这个不安全的指针通常是由攻击者构造的。例如,格式化串攻击中,常见的攻击方式是攻击者向自己精心构造的地址写入一个数值,这就存在不安全指针的解引用。Chen 等人[2]提到的针对 HTTP server-ghttpd 和 Wu-ftpd 的攻击实例,也是通过覆写指针,指针解引用后实施攻击。

根据这一特征,本文提出了指针污点分析方法,它是 DTA 的一种扩展,因此这一方法也分为三个步骤:a) taint source(标记污点来源);b) taint propagation(传递污染属性);c) taint sink(污点信息终点)。污点来源 source 就是来自外部的数据,可以是读取自文件、网络等;根据二进制指令的实际含义定义不同的传播规则(即 propagation);sink 是污点数据的到达处,通常会在此处设置安全规则检查。

下面定义两个重要的概念。

定义 1　污点标记(taint tag, T-tag)是内存数据的一个属性,表示该数据是否是来自外部、受污染的。真值表示是受污染的,假值表示不是受污染的。

定义 2　指针标记(pointer tag, P-tag)是内存数据的一个属性,表示该数据是否是合法指针,是否可以用做地址。真值表示是合法指针,假值表示是非法指针。

本文的防御方法就是围绕内存数据的这两个属性的标记、传播和检查。

2.1　标记污点来源

针对 T-tag,本文监控程序读取文件、网络数据等行为,将获取的数据初始化标记为受污染的,其他数据标记为非污染的。针对 P-tag,需要识别出该内存数据是否是合法的地址。有两种方式产生的数据可以用做指针,分别是动态分配空间的指针和静态分配空间的指针。初始状态下,只有通过这些方式产生的数据,才认为是合法指针,其他数据默认会标记为非法指针。

2.2　传递污染属性

数据作为不同指令类型操作数时,其传播规则也不同。表 1 概括了不同的传播策略。本文将指令主要分为算术运算指令、逻辑运算指令、数据传输指令和特殊指令。下面针对 T-tag 和 P-tag 分别进行详细描述。

表 1　污染属性传播策略

tag	instructions	propagation methods
T-tag	arithmetic instructions: mul op_1, op_2	$T(op_1) = T(op_1) \vee T(op_2)$
	data transfer instructions: mov op_1, op_2	$T(op_1) = T(op_2)$
	logical instructions: and op_1, op_2	$T(op_1) = T(op_1) \vee T(op_2)$
	special instructions: xor op_1, op_2	$T(op_1) = 0$
P-tag	add/subtract instructions: add op_1, op_2	$P(op_1) = P(op_1) \oplus P(op_2)$
	other arithmetic instructions: mul op_1, op_2	$P(op_1) = 0$
	and instructions(base address): and op_1, 0x11..00	$P(op_1) = P(op_1)$
	other logical instructions: or op_1, op_2	$P(op_1) = 0$
	data transfer instructions: mov op_1, op_2	$P(op_1) = P(op_2)$

2.2.1 T-tag 传播策略

算术运算指令:add、sub、mul 等

Example: mul op_1, op_2

Rule: $T(op_1) = T(op_1) \vee T(op_2)$

解析:只要两个操作数中任一个为受污染的,则污染属性传播到目标操作数中。

数据传输指令:mov 型指令,包括 mov、movsb 等

Example: mov op_1, op_2

Rule: $T(op_1) = T(op_2)$

解析:源操作数的污染属性直接传播到目标操作数中。

逻辑运算指令:and、or、shl 等

Example: and op_1, op_2

Rule: $T(op_1) = T(op_1) \vee T(op_2)$

解析:只要两个操作数中任一个为受污染的,则污染属性传播到目标操作数中。

特殊指令:xor、test 等

Example: xor op_1, op_1

Rule: $T(op_1) = 0$

解析:该指令的目的是将相应寄存器中的数据清空,因此污染属性将会清除。

2.2.2 P-tag 传播策略

算术指令(相加、相减):add、sub

Example: add op_1,op_2

Rule: $P(op_1) = P(op_1) \oplus P(op_2)$

解析:两个操作数都是合法指针,或者都不是合法指针,则目标操作数不是一个合法指针。这个很容易理解,两个地址之间加减,其结果是两者的相对偏移,所以不是一个合法的地址。如果两个操作数只有一个是合法指针,则这是一次通过基地址和偏移量计算地址的操作,其结果还是一个合法指针。

算术指令(相乘等其他指令):mul、div 等

Example: mul op_1, op_2

Rule: $P(op_1) = 0$

解析:不是正常的地址计算操作,因此其结果都标记为非法指针。

逻辑运算指令(取基地址的 AND 指令):and

Example: and op_1, 0x11..00

Rule: $P(op_1) = P(op_1)$

解析:这是一条比较特殊的 and 指令,目的是获取源操作数的基地址,因此其结果的 P-tag 属性与源操作数的 P-tag 属性相同。

逻辑运算指令(其他指令): or、not 等

Example: or op_1, op_2

Rule: $P(op_1) = 0$

解析:不是正常的地址计算操作,因此其结果都标记为非法指针。

数据传输指令:mov 型指令,包括 mov、movsb 等

Example: mov op_1, op_2

Rule: $P(op_1) = P(op_2)$

解析:源操作数的污染属性直接传播到目标操作数中。

2.3 攻击判定

攻击判定分为两类,一类是常见的控制数据攻击,监控跳转类指令(jump、call、ret 等),当其操作数的 T-tag 属性为真,即是来自外部的污点数据,表明这是一次攻击;另一类是监控指针的解引用:如果 T-tag 属性为假,即不是来自外部的污点数据,则不管 P-tag 的属性值,指针的解引用都是合法的;如果 T-tag 为真且 P-tag 为假,即当来自外部的污点数据,并且不是一个合法指针,被解引用时,则表明这是一次攻击;如果 T-tag 为真且 P-tag 也为真,虽然是来自外部的数据组成的地址,但是合法指针,所以可以解引用,这也很好地解决了之前防御方法存在的漏报误报问题[3,15]。表 2 定义了攻击判定的规则。

表 2 攻击判定规则

T-tag	P-tag	used as address
false	false	
false	true	
true	false	attack detection
true	true	

3 系统具体实现

本系统是在 Ubuntu Linux 下基于动态二进制分析框架 Pin 开发的指针污点分析工具(dynamic pointer taint analysis tool, DPTA)。Pin 可以在可执行二进制代码中插入一些探测函数,用于观察、记录、分析等。通过 Pin 提供的 API 可以编写各种分析工具,这样程序运行完以后,统计和分析结果也同时产生。DPTA 除了具有基本的 DTA 功能以外,主要着眼于指针污点分析,因此具有防御控制数据攻击和非控制数据攻击的能力。图 1 是 DPTA 系统框架。系统会监控系统调用,将来自外部的数据标记为 T-tag 和 P-tag 属性,并存储到 Tagmap 中。应用程序的每条指令通过 Pin 进行翻译后执行,同时 DPTA 会插入一些分析代码,实时监控程序运行状态。程序执行过程中会动态更新 Tagmap,并在安全敏感处执行检查,发现攻击时触发警报。

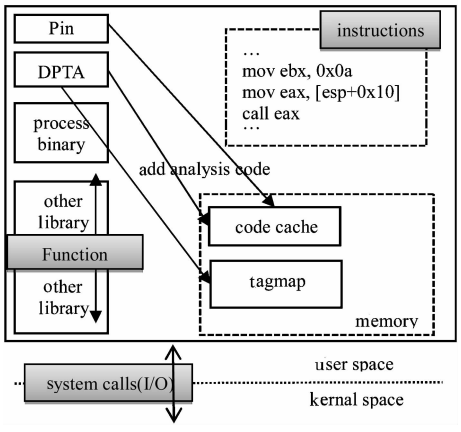

图 1 DPTA 系统框架

3.1 Tagmap 的设计

Tagmap,又被称为影子内存,是在系统中开辟的一块新区域,用来存储程序内存数据的标记信息。在本系统中,使用两个 bit 位来标志每一块内存的 T-tag 和 P-tag 属性,分别为 T-bit 和 P-bit。T-bit 为 0 表示不是污点数据,为 1 表示是污点数据。P-bit 为 0 表示不是一个合法指针,为 1 表示是一个合法指针。Tagmap 中标记的地址和实际的数据存储地址有一对映射关系,可以通过计算来获取。程序初始运行时,Tagmap 会根据实际情况初始化为不同的数值,默认下这两个 bit 位都是置为 0。T-bit 的初始化与传统的 DTA 相同,将 read、recv 等函数获取的数据标志为 1(污点数据)。P-bit 的初始化相对复杂,主要是

指针的产生可能会有很多种情况,因此指针识别是很重要的一环,下文会对其进行详述。识别出的指针,其 P-bit 将会被标志为 1(合法指针)。

3.2　指针识别

指针识别是本系统中非常重要的一部分,但是二进制中的指针识别向来是一件非常困难的事情。本文将指针区分为两类,即指向动态分配空间的指针和指向静态分配空间的指针,如全局变量的地址。图 2 展示了内存的布局,动态区域主要是堆和栈,静态区域包括静态数据区、代码区、动态链接库的静态数据区和代码区。下面分别对这两类指针的识别进行阐述。

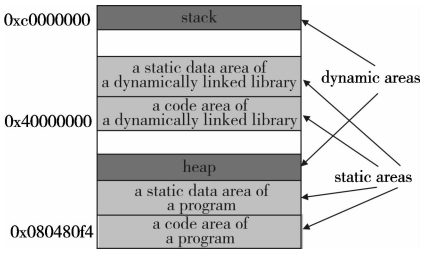

图 2　虚拟内存的空间分布

3.2.1　动态分配的堆空间指针

动态分配的空间一般需要一些系统调用,如 mmap、brk、mmap2、mremap、shmat 等。因此可以监控这一类系统调用,其返回的指针就是合法的堆空间指针。例如增加对系统调用 brk 的监视,根据 Pin 的 api,给 brk 绑定一个监控函数 post_brk_hook:

```
(void)syscall_set_post(&syscall_desc[_NR_brk],post_brk_hook)
```

当产生系统调用 brk 时,监控函数中会将返回的指针的 P-bit 设置为 1,表明是合法指针。同理,也可以添加对其他系统调用的监听。

栈空间的指针都是来自于栈指针(stack pointer),因此将程序开始时候的栈指针的 P-bit 设为 1。

3.2.2　静态分配空间的指针

静态分配空间的指针识别比较复杂。当程序被编译成重定向目标文件时,所有静态分配空间的引用都会放置在重定向表中。如果应用程序中有着完整的重定向表,可以精确地识别出所有静态的指针初始化。然而大部分情况下都缺少这些信息,因此对于这种情况,需要其他方法来识别指针。

DPTA 是基于 x86 指令集下的 Pin 开发的。通过反编译程序可知,这类指针初始化具有一个特征,都是通过类似 movl 的指令,将一个 32-bit 的常数赋值给寄存器,据此可以保守地识别出 x86 下的静态分配空间指针。

本文首先获得静态空间的起始地址和终止地址,然后在 DPTA 插桩的时候检查 movl 类的指令。如果其操作数是立即数和寄存器,并且立即数在静态空间的地址范围内,则认为其是合法的指针初始化。这些操作虽然比较复杂,但是这是插桩代码,因此其对效率影响有限。

3.3　污染属性传播和攻击判定

污染属性传播也就是在程序运行中,每块内存数据所对应的两个 bit 位的变化。按照上文提到的传播策略,当程序运行时,Tagmap 中的数据将按照表 1 中的策略进行更新。

攻击判定,首先针对跳转类指令(jump、call、ret 等)进行判断,这与传统的 DTA 相同,此处不作详述。针对指针解引用的判断,根据指针解引用指令(如 mov 指令)的操作数,如果源操作数是指针,T-bit 为 1 且其 P-bit 为 0,表明要读取一个非法指针指向的内存区域数据,判断这是一次对非法指针的解引用,是一次攻击。如果目的操作数是指针, T-bit 为 1 并且其 P-bit 为 0,表明将对一个非法指针指向的内存区域写入数据,判断其为攻击。代码 3 是一段攻击判定的示例代码。

代码 3　攻击判定示例代码

```
if (INS_OperandIsMemory(ins, OP_1))
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)alert_mov,
IARG_FAST_ANALYSIS_CALL,IARG_INST_PTR,IARG_END);
else if (INS_OperandIsMemory(ins, OP_2))
    INS_InsertCall(ins,IPOINT_BEFORE, (AFUNPTR)alert_mov,
IARG_FAST_ANALYSIS_CALL,IARG_INST_PTR,IARG_END);
```

其中:alert_mov 方法将会按照本文定义的策略对该内存地址的 T-bit 和 P-bit 进行检查,判断是否是一次攻击。

4　实验评估

在 Ubuntu Linux 下对 DPTA 进行实验评估,内核版本是 3.4,运行的平台为 CPU 2.67 GHz,内存 4 GB。实验一方面验证该工具是否能够检测出非控制数据攻击和控制数据攻击,另一方面测试工具对应用程序效率的影响。

4.1　攻击检测验证

本文选取了具有代表性的四个控制数据攻击和四个非控制数据攻击,进行了多次测试。表 3 的测试结果表明 DPTA 针对控制数据攻击和大部分非控制数据攻击都具有良好的防御能力,但是对于不通过指针修改关键数据的攻击还无法防御,这是本文系统的局限性。

表 3　DPTA 攻击防御测试

program	vulnerability	detected
Polymorph[17]	stack overflow	yes
Atphttpd[18]	stack overflow	yes
Traceroute[19]	double free	yes
Sendmail[20]	BSS overflow	yes
Wu-ftpd[2]	format string attack against user identity data	yes
Null httpd[2]	heap corruption attacks against configuration data	yes
Ghttpd[2]	stack buffer overflow attack against user input data	yes
OpenSSH[2]	integer overflow attack against decision-making data	no

4.2　性能测试

本文选取了五个程序来测试系统的性能损耗,同时使用了 Kemerlis 开发的 DTA 工具 Libdft[16] 进行参照,表 4 给出了最终的测试结果。总体而言,系统的损耗从 1.5x 到 28.0x 不等,平均性能损耗是 12.5x,这与 Memcheck 的运行负载相近。相比于 Libdft,DPTA 的系统负载比较高,这是因为采取了比较严格的监测策略,除了监控跳转指令外,还包括了非常常见的数据传送指令(mov 等),这会给系统增加非常大的运行开销。但是本文系统可以防御大部分的非控制数据攻击,而 Libdft 没有这一能力。

表 4　DPTA 性能测试

program	benchmark	time/s	libdft	DPTA
tar-1.26	archive 160 MB data	1.543	1.2x	6.5x
gzip-1.3	Decompress 20 MB data	1.253	4.3x	28.0x
bzip2-1.0.6	compress 20 MB data	3.813	6.1x	21.4x
grep-2.5.4	find pattern in 160 MB file	3.335	2.9x	5.2x
ccrypt-2.5	encrypt 16 MB data	4.681	1.2x	1.5x

5 结束语

本文在动态污点分析的基础上,针对非控制数据攻击提出了一个改进的指针污点分析方法,通过跟踪内存数据的污点标记和指针标记,监控是否存在非法的指针解引用,实现了原型工具 DPTA。实验评估表明,该工具可以防御控制数据攻击和大部分非控制数据攻击。虽然 DPTA 实现了本文防御模型,但是还存在一些问题需要进一步研究:

a)实验表明本文原型系统运行效率并不是很理想,距离商业化应用还有距离。笔者计划通过两方面进行优化:一方面优化 Tagmap 的数据结构,更新 Tagmap 信息是系统中常见的操作,因此快速获取 Tagmap 中的信息会对效率产生很大影响;另一方面手动分析程序调用的常用函数的语义信息,获取其是否涉及污点传播,对于不涉及污点传播的函数不进行插桩分析。

b)目前采取的指针识别方法并不是很准确,尤其是静态分配空间的指针,下一步计划分析识别失败的指针,归纳其特征,提高指针识别的准确率。

c)本系统无法检测出不通过覆写指针实施的非控制数据攻击,可以通过结合静态分析的方法防御,但这不是下一步的主要方向。

参考文献:

[1] ROEMER R, BUCHANAN E, SHACHAM H, *et al*. Return-oriented programming: systems, languages, and applications [J]. ACM Trans on Information and System Security, 2012, 15(1): 2.

[2] CHEN Shuo, XU Jun, SEZER E C, *et al*. Non-control-data attacks are realistic threats[C]//Proc of the 14th Conference on USENIX Security Symposium. Berkeley: USENIX Association, 2005: 12.

[3] SLOWINSKA J M. Using information flow tracking to protect legacy binaries[M]. [S. l.]: Vrije Universiteit, 2012.

[4] NEWSOME J, SONG D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software [EB/OL]. (2005). http://valgrind.org/docs/newsome2005.pdf.

[5] EGELE M, SCHOLTE T, KIRDA E, *et al*. A survey on automated dynamic malware-analysis techniques and tools[J]. ACM Computing Surveys, 2012, 44(2): 6.

[6] 王蕊,冯登国,杨轶,等.基于语义的恶意代码行为特征提取及检测方法[J].软件学报,2012,23(2):378-393.

[7] 刘杰,王嘉捷,欧阳永基,等.基于污点指针的二进制代码缺陷检测[J].计算机工程,2012,38(24):46-49.

[8] LUK C K, COHN R, MUTH R, *et al*. Pin: building customized program analysis tools with dynamic instrumentation[C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2005: 190-200.

[9] SCHLESINGER C, PATTABIRAMAN K, SWAMY N, *et al*. Modular protections against non-control data attacks[C]//Proc of the 24th IEEE Computer Security Foundations Symposium. 2011: 131-145.

[10] Van ACKER S, NIKIFORAKIS N, PHILIPPAERT P, *et al*. Valueguard: protection of native applications against data-only buffer overflows[M]//Information Systems Security. Berlin: Springer, 2011: 156-170.

[11] BHATKAR S, SEKAR R. Data space randomization[M]//Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2008: 1-22.

[12] XU Jun, NAKKA N. Defeating memory corruption attacks via pointer taintedness detection[C]//Proc of International Conference on Dependable Systems and Networks. Washington DC: IEEE Computer Socitey, 2005: 378-387.

[13] DALTON M, KANNAN H, KOZYRAKIS C. Real-world buffer overflow protection for userspace & kernelspace[C]//Proc of the 17th Conference on Security Symposium. [S. l.]: USENIX Association, 2008: 395-410.

[14] 汪洁,杨柳.基于蜜罐的入侵检测系统的设计与实现[J].计算机应用研究,2012,29(2):667-671.

[15] SLOWINSKA A, BOS H. Pointer tainting still pointless: (but we all see the point of tainting)[J]. ACM SIGOPS Operating Systems Review, 2010, 44(3): 88-92.

[16] KEMERLIS V P, PORTOKALIDIS G, JEE K, *et al*. Libdft: practical dynamic data flow tracking for commodity systems[C]//Proc of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments. [S. l.]: ACM Press, 2012: 121-132.

[17] Polymorph filename buffer overflow vulnerability[EB/OL]. (2003). http://www.securityfocus.com/bid/7663.

[18] ATPHTTPD buffer overflow exploit code[EB/OL]. (2001). http://www.securiteam.com/exploits/6B00K003GY.html.

[19] SAVOLA P. LBNL traceroute heap corruption vulnerability [EB/OL]. (2000). http://www.securityfocus.com/bid/1739.

[20] DOWD M. Sendmail header processing buffer overflow vulnerability [EB/OL]. http://www.securityfocus.com/bid/6991.

(上接第 3761 页)

[5] YANG Wu, MA Xing-guo, WANG Wei, *et al*. Research on reputation evaluating model for WSN nodes based on vertical and horizontal analysis[C]//Proc of the 2nd International Conference on Networks Security, Wireless Communications and Trusted Computing. Washington DC: EE Computer Society, 2010: 293-296.

[6] 杨光,印桂生,杨武,等.无线传感器网络基于节点行为的信誉评测模型[J].通信学报,2009,30(12):18-26.

[7] 姚红燕,周鸣争,刘涛.一种基于节点行为的无线传感器网络信誉计算模型[J].南通大学学报:自然科学版,2011,10(1):5-9.

[8] GANERIWAL S, BALZANO L K, SRIVASTAVA M B. Reputation-based framework for high integrity sensor networks[J]. ACM Trans on Sensor Networks, 2008, 4(3): 15.

[9] HEINZELMAN W R, CHANDRAKASAN A P, BALAKRISHNAN H. An application specific protocol architecture for wireless microsensor networks [J]. IEEE Trans on Wireless Communications, 2002, 1(4): 660-670.

[10] JSANG A, ISMAIL R. The beta reputation system[C]//Proc of the 15th Bled Electronic Commerce Conference. 2002: 41-55.

[11] 刘念,刘孙俊,刘勇,等.一种基于免疫的网络安全态势感知方法[J].计算机科学,2010,37(1):126-129.

[12] KARLOF C, WAGNER D. Secure routing in wireless sensor networks: attacks and counter measures[J]. Ad hoc Networks, 2003, 1(2): 293-315.