# 基于访问控制的动态着色技术在攻击检测中的研究

王 磊 茅 兵 谢 立
(南京大学软件新技术国家重点实验室 南京 210093) (南京大学计算机科学与技术系 南京 210093)

**摘 要** 内存腐烂攻击在软件安全攻击中占据着较大的比重。近来,动态着色技术得到了越来越多的关注,这种技术通过在访问内存时检测指针的完整性来抵御攻击。然而,存在一类可以绕过指针完整性检查的策略来进行攻击的实例,比如数组的越界访问攻击。提出了一种基于动态着色跟踪分析的方法来解决这类已有着色技术不能检测的问题。其思想是,借助于内存访问控制的思路,首先像已有的动态着色技术那样,在内存访问时对指针进行完整性检查,然后检查指针将要访问的内存区域是否处于指针合理的访问范围之内。原型系统是基于 Valgrind 的,并不需要源码,因此可以用于很多商业软件。初步实验验证结果表明,该方法可以有效地检测出很多类型的攻击,系统的性能损耗接近于 Memcheck 这种常用的内存错误检测工具。

**关键词** 内存腐烂攻击,指针着色,内存访问

## Memory Corruption Detection Based on Dynamic Taint Analysis and Access Control

WANG Lei MAO Bing XIE Li
(State Key Laboratory of Computer Software and New Technology, Nanjing University, Nanjing 210093, China)
(Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

**Abstract** Memory corruption attacks account for most parts of malicious attacks toward software security. Recently dynamic taint analysis was proposed and was gaining momentum. This proposed technique attempts to defeat attacks by checking the taintedness and integrity of pointers when accessing memory. Unfortunately, there exists some class of attacks without tainting pointers, such as array bounds violation attacks using pointers. We proposed a novel approach to defeat this kind of undetected attacks using taint-based tracking analysis. Our notion is based on the memory access control, that is, first, we will check the taintedness of the pointers when accessing memory like existing taint-based approaches, second, we will check whether or not the memory area pointed by the pointer is in the legitimate range of the accessing pointer. Our implementation does not need source code and is based on Valgrind, hence works on commodity software. To demonstrate our idea, we performed a preliminary empirical experiments, the results are quite promising: our system can effectively detect a wide range of attacks, and the average runtime overhead is close to Memcheck, a widely used memory error detector.

**Keywords** Memory corruption attacks, Taint pointers, Memory access

## 1 引言

由于现代软件的日益复杂化,出现了越来越多的软件漏洞。US-CERT[1]发布的统计数据表明,近年来,软件的安全漏洞数量一直维持在一个很高的水平。通过触发软件安全漏洞,攻击者可以非法执行任意恶意代码,甚至控制整个系统,从而带来无法预料的严重后果。

近来,动态着色跟踪分析技术(动态信息流监控)得到了广泛的关注,被认为是一种很有前景的攻击检测平台。动态着色分析技术的思想很直观,它将外来的数据着色,跟踪着色数据在程序中的运行,并把与外来着色数据具有传播关系的数据同样着色,最后在安全敏感操作(函数返回地址、指针操作)处进行检查,如果有着色的数据在那些地方被使用,就被触发一个攻击异常。

现在已有很多基于动态着色技术的方法,主要可以分为两类:基于硬件的和基于软件的。基于硬件的着色技术需要特殊的操作系统和处理器的支持[2,12,17,20,33,39],它们的优势在于较低的性能损耗和透明的着色分析支持;缺点在于只支持简单的着色跟踪策略,会导致大量的漏报。基于软件的着色技术[14,16,25,26,27,34,36]需要借助动态二进制代码转换工具,如PIN[3],Valgrind[30,31],虽然这种方法将会带来比较大的性能损耗,但是其动态着色跟踪策略和检测策略十分灵活,使得它们很容易适应各种攻击检测手段。因此,我们的方法是基于软件的动态着色分析技术的。

到稿日期:2009-02-10 返修日期:2009-04-06 本文受国家自然科学基金(60773171,90818022,60721002),国家863高技术计划(2007AA01Z448),国家973重点基础研究计划(2009CB320705),江苏省自然科学基金(BK2007136)资助。

王 磊 硕士研究生,主要研究方向为软件安全,E-mail:brytwang@gmail.com;茅 兵 教授,博士生导师,主要研究方向为软件安全、安全操作系统;谢 立 教授,博士生导师,主要研究方向为信息安全、分布式系统。

目前，几乎所有的低级软件漏洞攻击，比如栈溢出攻击、格式化字符串攻击，其手段都有一个共同的特征，就是必须着色指针，然后引用被着色的指针，成功触发一个恶意攻击。所以，指针是安全敏感数据。确保指针的完整性，是解决这些攻击的最佳切入点。这个想法已得到很多相关工作的支持[18,20,21]。

进一步研究发现，仅仅保护指针的完整性是不够的。因为存在一类可以绕过指针完整性保护方法的攻击，比如数组的越界访问攻击。这类攻击方法的本质在于诱使指针访问不在它合法区域中的内存，而不是着色修改指针本身。

我们提出了一个增强型的基于传统指针着色概念的方法。使用程序内外结合着色的技术来实现我们的思想。程序内外共同着色的思想与文献[20,28,29]中的类似。首先着色程序内部数据，使得指针及其所能访问的合法区域打上相同的标签。因此，像访问控制那样，可以保证指针只能访问其合法的内存区域(指针和指针访问的区域的标签是相同的)。同时，与传统的基于着色技术一样，将外来输入的数据打上标签，标记为不可信任，并且监控它们的运行。由于指针是安全敏感数据，故在使用指针访问内存时，将首先检测指针本身是否被着色。如果被着色，立刻报错；否则，将进一步比较指针欲访问的内存的标签和此指针本身的标签。如果标签相同，则该内存访问行为是合法的，否则就发生越界访问错误。

系统原型是基于动态二进制转换工具 Valgrind 上的，定义在二进制级别。虽然定义在二进制级别，将会失去很多信息，但该方法仍然可以有效地检测出很多攻击，特别是检测与动态分配内存相关的攻击，比如栈溢出攻击。

## 2 思路的出发点

US-CERT[1]的统计数据表明，在软件漏洞攻击中，低级的漏洞攻击占据了绝大部分。低级漏洞攻击通常包括缓冲区溢出攻击、格式化字符串攻击和堆溢出攻击。这些攻击的共同特征都来自被外来数据着色过的指针。但是，我们发现，没有着色指针，即在保持指针完整性的基础上，同样可以发生攻击。

图 1 和图 2 就是两个很好的绕过传统指针完整性保护检测手段的攻击实例。

```
void bounds_attack()
{
  1. int len = 10;
  2. char buffer[20];
  3. int num[10];
  4. int i;
  5. ……
  6. scanf(buffer);
  7. for (i=0;i<len;i++)
  8. printf("%d", *(num+i));
}
```

图 1 使用未着色的指针进行的越界访问

```
void dynamic_attack(n)
{
  1. int l;
  2. char *buffer;
  3. buffer=(char *)malloc(n);
  4. if (buffer == NULL) return;
  5. for (i=0;i<=n;i++)
  6. *(buffer+i)='a';
  7. printf(buffer);
  8. free(buffer);
}
```

图 2 使用未着色的指针进行的非法内存访问

在图 1 中，第 6 行，函数 scanf()可以读入超过 20 字节的数据到 buffer，覆盖邻近变量 len。在第 7 行，len 间接控制指针 num，因此攻击者可以诱使指针 num 读取在 buffer 中的外来数据，于是产生一个越界访问错误。在这个例子中，没有指针被外来数据着色，所以传统的基于指针完整性保护的方法无法检测攻击。在图 2 中，第 6 行，指针 pointer 同样没有被着色，但是当 $i$ 等于 $n$ 时，内存访问操作是个越界错误。

因此，仅仅检测指针的完整性是不够的，必须限制指针，使指针在访问时引入访问控制机制，即每个指针只能访问其合法的内存区域范围。因此在指针访问内存时，必须检测要访问的内存区域是否在该指针的合法内存访问区域中。

## 3 设计和实现

思路的具体实现，是基于动态二进制转换工具 Valgrind 的。

图 3 是系统的大致组成框架。输入是一个在二进制级别的原始程序，它在我们的系统中将会在运行时进行二进制动态转换。我们的原型系统主要由两部分组成：一部分是进行外部和内部数据的动态着色，另一部分是通过指针访问内存时进行安全检测。

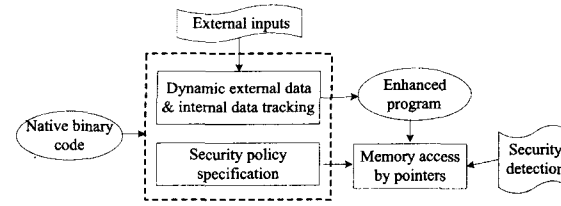


图 3 基于动态着色分析技术的原型系统

已经提到，限制每个指针，使其只能访问其合法内存区域是必须的，因此系统对程序内部的指针及其指向的内存区域打上相同的标签。通过标签识别的方法，可以判别哪个指针可以访问哪片内存区域，因此通过标签比较的方法来对指针进行访问控制。像传统的动态着色技术一样，将外来的输入数据打上标签，跟踪它在程序中的运行来正确地传播它的标签。当使用指针进行内存访问时，将触发安全检测策略，检测指针的完整性，检测指针及其将要访问的内存区域两者的标签。

### 3.1 系统的具体实现

指针是安全敏感数据，每个指针都被限定在其合法的内存访问区域，因此将在程序内部的指针及其对应的内存区域进行着色，这就是所谓的内部着色。像原来的着色技术那样，把外来的数据进行着色，这就是所谓的外部着色。监控整个程序的运行，来合理而准确地进行着色标签的传播。最后，当通过指针进行内存访问时，安全检测策略将会检测通过该指针进行内存访问是否是合法的。

下面将详细介绍系统原型的实现。

### 3.2 指针识别

准确地进行指针识别，是一个不可判定的问题，特别针对二进制级别的程序相当困难[11,15,24,35,37,38]。主要的困难在于识别静态分配的指针，因为在二进制级别，很多语意信息已经无法获得。因此，在系统中使用比较保守的指针识别方法。

· 指向静态分配内存的指针

该思路的最大挑战就是识别指向静态分配内存的指针，因为在二进制级别提供的语意非常有限。在 ELF 文件的可重定位表中，会标记出所有指向静态区域的指针信息，但是可重定位表在商业软件中并不总是可得的。因此，只能采取保守的方法来识别指向静态分配内存区域的指针。Valgrind 中的指令集是类 RISC 结构的，系统就在 Valgrind 的中间代码 VEX 上进行指针的识别。指针有一个最显著的特点，就是在使用前必须先进行初始化操作。因此借助这个特点，在 Valgrind 的中间代码 VEX 上可以进行识别操作。其识别的思路如下：指针的初始化过程中 Valgrind 的中间代码的对应形式是两条连续的指令 Add32 和 STle。基于识别这两条连续的指令，就可以比较保守地识别出指向静态分配内存区域的指针，并给这些指针打上标签。

· 指向动态分配内存的指针

通常需要显示地调用相关内存分配函数（如 malloc，calloc）进行内存的动态分配。为了限制动态生成的指针，可截获这些动态内存分配函数，把动态分配得到的内存区域打上标签来着色，并用相同的标签给指向这些内存区域的指针进行着色。所以，使用标签来对动态生成的指针进行访问控制，指针只能访问和它有相同标签的内存区域。同样，可调用截获函数来对函数返回地址进行动态着色。

### 3.3 外来输入数据的着色

软件遭受恶意攻击，通常都是源于外部恶意的数据输入。因此，将对外来（网络、键盘、文件系统）的数据进行着色。与传统的动态着色技术一样，需要截获数据输入函数（read，scanf）。

### 3.4 标签的传播

系统需要监控每条数据操作指令，以此来决定目标操作数是否需要着色。在 Valgrind 的中间代码中，指令集可以大致分为两大类，因此在基于这两大类指令集的基础上，给出了一个比较准确的标签传播策略。

· 数据移动指令

在 Valgrind 中，指令 LOAD，STORE 属于数据的转移指令。针对此类指令的标签传播策略非常简单。如果源数据是着色的，那么目标数据也将被着色，并打上和源数据相同的标签。

· 算术运算指令

在 Vlagrind 中的 ADD，SUB 等指令就属于此类指令。在该类指令中，标签的传播策略如下：如果操作数中有一个或多个是着色的，那么操作结果也是着色的。虽然这些指令将会影响条件状态寄存器，在这里忽略了条件状态寄存器的着色行为。由于条件状态寄存器会频繁地受影响，如果将它考虑进去，所带来的性能损耗将是巨大的。

### 3.5 标签检测

这一步就是用来检测指针的内存访问行为是否是合法的。如果基于指针的内存访问是非法的，那么立刻报错。所以，需要截获每次指针的内存访问行为，无论它是读操作还是写操作。在使用指针进行内存访问时，首先对指针本身进行完整性检查。如果指针已被外来数据着色，像传统的基于指针完整性的方法那样，立刻发出异常处理；否则，进一步检查被外来数据着色的指针所要访问的内存区域是否是合法的，通过比较指针本身的标签值和此指针将要访问的内存区域的标签值来进行判断。如果两个标签值不同，此内存访问操作是非法的，否则就是合法的内存访问。在 Valgrind 中，当用指针进行内存访问时，将会有两次连续的 LOAD 操作，我们会比较第一次 LOAD 的标签值和第二次 LOAD 对应的标签值。

## 4 实验检验

对系统进行了初步的实验测试。实验的主要目的是验证系统针对恶意攻击的有效性检测以及系统运行的性能损耗。所有的实验都在 Debian 系统下进行，内核版本是 2.6.18，运行的平台是 2.4GHz Intel Xeon CPU，1G 的 RAM。

### 4.1 攻击检测验证

我们选取了一个比较流行的攻击测试平台 Bugbench[40]。从 Bugbench 中选取了 5 个常用的有漏洞的程序。这 5 个程序分别包含几个漏洞，因此只是选取这些程序中的一个比较常见的漏洞来进行测试。

表 1 中列出了本系统针对这 5 个漏洞程序的测试结果。对每个程序重复测试了 10 次，都得到了一致的测试结果。表 1 中的结果表明，系统具有强大的攻击检测能力。

表 1 漏洞程序的攻击检测

| Application | Vulnerability | Attack type | Detected |
|---|---|---|---|
| bc-1.06 | more_arrays(storage:153) | heap buffer overflow | yes |
| gzip-1.2.4 | get_istat(gzip.c:816) | array overflow | yes |
| man-1.5h1 | get_section_list (man.c:960) | stack arra y overflow | yes |
| ncompress-4.2.4 | strcpy(compress42.c:896) | stack array overflow | yes |
| polymorph-0.4.0 | convert_fileName (polymorph.c:181) | stack array overflow | yes |

### 4.2 性能测试

使用了 7 个程序来测试我们系统的性能损耗。对每一个测试程序，分别将它们运行在原始状态、Memcheck 和我们的系统下。Memcheck 是一个在 Valgrind 中广泛使用的插件，主要用来测试内存访问错误。

表 2 给出了测试的性能损耗。对于 Apache 服务器程序，使用了 Web 测试工具 ab，发送的请求次数是 3,000，并发次数是 100，向 Apache 服务器发送请求；对于 ProFTPD 程序，使用了开源工具 dkftpbench 进行测试，模拟 10,000 个用户来重复进行登录。

表 2 系统性能损耗分析比较

| Program | Benchmark | Time(s) | Memcheck | Our system |
|---|---|---|---|---|
| tar-1.16 | archive 141 MB data | 1.675 | 5.2X | 6.0X |
| gzip-1.3.5 | decompress 15MB data | 1.143 | 22.8X | 30.0X |
| bzip2-1.0.5 | compress 14MB data | 4.693 | 20.0X | 22.3X |

| | | | | |
|---|---|---|---|---|
| grep -2.5.1 | find pattern in 141MB file | 3.684 | 5.2X | 5.6X |
| apache -2.0.40 | ab | 10.656 | 1.1X | 1.2X |
| ccrypt-1.7 | encrypt 15MB data | 5.370 | 7.1X | 7.8X |
| proftp -1.2.10 | dkftpbench | 3.255 | 11.0X | 13.3X |
| average | | 4.354 | 7.7X | 8.6X |

从表2中可以看到，总体而言，系统的平均性能损耗是9倍，接近于Memcheck。

## 5 相关工作

动态着色跟踪分析实际上是程序运行监控的一种变形。目前，已有大量针对程序运行监控的研究工作，比如栈溢出监控[4,23]、指针分析[32,24]、边界检查[6,5]、地址空间随机化[7,8]和控制流监控[9,10]，都是通过监控程序运行中不同的对象来检测恶意攻击的。虽然动态着色分析技术有其本身固有的不足[22]，但是它依然有独特的优势前景。

LIFT[13]使用静态分析和加速指令集来降低性能损耗。它通过静态分析，省略一些不必要的信息流监控路径，合并许多类似的标签检查操作，这样可以有效地在目标程序和转换代码中进行切换。在系统中没有考虑不必要的信息监控路径，所以性能损耗要高于LIFT。

在文献[11]中提供了一种减少性能损耗的方法。使用函数调用间的静态分析，删除不必要的监控路径，将原始的程序编译转化成安全增强型的程序。该方法是基于动态数据流和静态数据流基础上的。我们的方法只考虑了动态的二进制分析。

Dytan[19]提供了非常灵活的策略，可以定制哪个数据需要被着色、标签在程序中如何传播、在何处进行标签的检测。更重要的是，它提供了两种标签传播机制，分别是数据流传播、数据流和控制流传播。目前本系统只考虑了数据流传播，将来会涉及控制流的标签传播。

文献[29]中的工作使用了程序内部数据的着色方法来检测非法内存访问。其出发点是，当动态内存分配时，为分配得到的内存区域和返回的指针打上相同的标签。当使用指针访问内存时，通过指针本身的标签和欲访问的内存区域的标签的比较来判定该访问操作的合法性。采用的是限制每个指针，使其只能访问其合法内存区域的思路，与文献[29]类似。但是文献[29]的工作只是针对内存非法访问的，因此它无法检测其它类别的攻击，比如格式化字符串攻击。

**结束语** 本文在基于指针完整性和指针访问控制的基础上，提出了一种新的动态着色分析技术。当在指针访问内存时，首先检测指针本身的完整性，如果指针已被着色，像现有的着色分析方法那样，认为这是一次攻击行为；否则，继续检测指针欲访问的内存区域是否在该指针的合法访问区域之中。通过比较指针本身的标签和欲访问内存的标签来进行判定。根据上述思想，我们设计了一个原型系统，详细阐述了该系统的设计、实现和实验验证。初步的实验表明，该系统可以检测绝大多数攻击实例，并且性能损耗也是比较乐观的。

## 参 考 文 献


[1] http://cert.org/. January 2009

[2] Venkataramani G, Doudalis I, Solihin Y, et al. FlexiTaint: A Programmable Accelerator for Dynamic Taint Propagation[C]// Proceedings of the 14th International Symposium on High Performance Computer Architecture. 2008

[3] Luk C, Cohn R, Muth R, et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation[C]// Proceedings of the 2005 ACM SIFPLAN Conference on Programming Language Design and Implementation. 2005

[4] Cowan, Pu C, Maier D, et al. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks[C]// Proceedings of the 7th USENIX Security Symposium. 1998

[5] Dhurjati D, Adve V. Backwards-compatible array bounds checking for C with very low overhead[C]// Proceedings of the 28th International Conference on Software Engineering. 2006

[6] Nethercote N, Fizhardinge J. Bounds-Checking Entire Programs without Recompiling[C]// Proceedings of the Second Workshop on Semantics, Program Analysis, and Computering Environments for Memory Management(SPACE 2004). 2004

[7] Shacham H, Page M, Pafaff B, et al. On the effectiveness of address space randomization[C]// ACM Computer and Communication Security Symposium. 2004

[8] Bhatkar S, DuVarney D, Sekar R. Address obfuscation: An efficient approach to combat a broad range of memory error exploits [C]// USENIX Security Symposium. 2003

[9] Abadi M, Budiu M, Erlingsson U, et al. Control-Flow-Integrity: Principles, Implementations, and Applications[C]// ACM Conference on Computer and Communication Security. 2005

[10] Kiriansky V, Bruening D, Amarasinghe S. Secure execution via program shepherding[C]// USENIX Security. 2002

[11] Chang W, Streiff B, Lin C. Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis[C]// Proceedings of the 15th ACM Conference on Computer and Communication Security. 2008

[12] Dalton M, Kannan H, Kozyrakis C. Raksha: A Flexible Information Flow Architecture for Software Security[C]// Proceedings of the 34th International Symposium on Computer Architecture. 2007

[13] Qin F, Wang C, Li Z, et al. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks [C]// Proceedings of the 39th the Intl. Symposium on Microarchitecture. 2006

[14] Xu W, Bhatkar S, Sekar R. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks[C]// Proceedings of the 15th USENIX Security Conference. 2006

[15] Katsunuma S, Kurita H, Shioya R, et al. Base Address Recognition with Data Flow Tracking for Injection Attack Detection[C] // Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing. 2006

[16] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C]// Network and Distributed System Security Symposium(NDSS). 2005

[17] Crandall J, Chong F. Minos: Architectural support for software security through control data integrity[C]// Proceedings the 37th International Symposium on Microarchitecture. 2004

[18] Kong J, Zhou C, Zhou H. Improving Software Security via Runtime Instruction-level Taint Checking[C]// Proceedings of the

1st Workshop on Architectural and System Support for Improving Software Dependability. 2006

[19] Clause J, Li W, Orso A. Dytan: A Generic Dynamic Taint Analysis Framework[C]// Proceedings of the International Symposium on Software Testing and Analysis. 2007

[20] Chen S, Xu J, Nakka N, et al. Defeating memory corruption attacks via pointer taintedness detection[C]// IEEE International Conference on Dependable Systems and Networks(DSN). 2005

[21] Chen S, Pattabiraman K, Kalbarczyk Z, et al. Formal Reasoning of Various Categories of Widely Exploited Security Vulnerabilities Using Pointer Taintedness Semantics[C]// 19th IFIP International Information Security Conference(SEC2004). 2004

[22] Cavallaro L, Saxena P, Sekar R. On the Limits of Information Flow Techniques for Malware Analysis and Containment[C]// Proceedings of the GI SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). 2008

[23] Ruwase O, Lam M. A practical dynamic buffer overflow detector [C]// Proceedings of the Network and Distributed System Security Symposium. 2004

[24] Avots D, Dalton M, Livshits B, et al. Improving Software Security with a C Pointer Analysis[C]// Proceedings of the 27th International Conference on Software Engineering. 2005

[25] Egele M, Kruegel C, Kirda E. Dynamic Spyware Analysis[C]// Proceedings of the 2007 USENIX Annual Conference(Usenix'07). 2007

[26] Moser A, Kruegel C, Kirda E. Exploiting multiple execution paths for malware analysis[C]// Proceedings of the 2007 IEEE Symposium on Security and Privacy(Oakland'07). 2007

[27] Vogt P, Nentwich F, Jovanovic N, et al. Cross - Site Scripting Prevention with Dynamic Data Tainting and Static Analysis[C]// Proceedings of the Network and Distributed System Security Symposium(NDSS'07). 2007

[28] Akritidi P, Cadar C, Raiciu C, et al. Preventing memory error exploits with WIT[C]// Proceedings of 2008 IEEE Symposium on Security and Privacy(Oakland'08). 2008

[29] Clause J, Doudails I, Orso A, et al. Effective Memory protection Using Dynamic Tainting[C]// International Conference on Automated Software Engineering. 2007

[30] Nethercote N, Seward J. Valgrind: A framework for heavy - weight dynamic binary instrumentation[C]// Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation(PLDI'07). 2007

[31] Seward J, Nethercote N. Using Valgrind to detect undefined value errors with bit-precision[C]// ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference. 2005

[32] Cowan C, Beattie S, Johansen J, et al. PointerGuard: Protecting Pointers From Buffer Overflow Vulnerabilities[C]// Proceedings of the 12th USENIX Security Symposium. 2003

[33] Suh G, Lee J, Zhang D, et al. Secure Program Execution via Dynamic Information Flow Tracking[C]// Proceedings of the 11th Conference on Architectural Support for Programming Languages and Operating Systems. 2004

[34] Lam L, Chiueh T. A general dynamic information flow tracking framework for security applications[C]// Proceedings of the 22nd Annual Computer Security Applications Conference. 2006

[35] Dalton M, Kannan H, Kozyrakis C. Real-World Buffer Overflow Protection for User and Kernel Space[C]// Proceedings of the 17th USENIX Security Symposium. 2008

[36] Haalfond W, Orso A, Manolios P. Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks[C]// Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2006

[37] Hind M, Pioli A. Which pointer analysis should I use? [C]// Proceedings of the International Symposium Testing and Analysis. 2000

[38] Hind M. Pointer Analysis: Haven't We Solved This Problem Yet? [C]// Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 2001

[39] Vachharajani N, Bridges M, Chang J, et al. RIFLE: An Architectural Framework for User-Centric Information-Flow Security [C]// Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture. 2004

[40] Lu S, Li Z, Qin F, et al. Bugbench: Benchmarks for evaluating bug detection tools[C]// Proceedings of the workshop on the Evaluation of Software Defect Detection tools. 2005

(上接第 82 页)

[2] 诸葛建伟,韩心慧,叶志远,等. 基于扩展目标规划图的网络攻击规划识别算法[J]. 计算机学报,2006,29(8):1356-1366

[3] Cuppens F, Autrel F, Miege A, et al. Recognizing Malicious Intention in an Intrusion Detection Process[C]// Second International Conference on Hybrid Intelligent Systems. Santiago, 2002

[4] Qin X, Lee W. Attack Plan Recognition and Prediction Using Causal Networks[C]// Proceedings of the 20th Annual Computer Security Applications Conference. 2004: 370-378

[5] Ning P, Xu D. Learning Attack Strategies from Intrusion Alerts [C]// Proceedings of the 10th ACM conference on computer and communications security. Washington D. C., USA, 2003: 200-209

[6] Huang MY, Wicks TM. A large-scale distributed intrusion detection framework based on attack strategy analysis[J]. Computer Netwroks, 1999: 2465-2475

[7] 鲍旭华,戴英侠,冯萍慧,等. 基于入侵意图的复合攻击检测和预测算法[J]. 软件学报,2005,16(12):2132-2138

[8] Youn S, Oh K. Intention Recognition using a Graph Representation[C]// Proceedings of World Academy of Science and Technology. 2007(21): 13-18

[9] Breazeal C. Social interactions in HRI: the robot view[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2004, 34(2): 181-186

[10] Blaylock N, Allen J. Fast Hierarchical Goal Schema Recognition [C]// Proceedings of AAAI. 2006: 8-15

[11] Pearl J. Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference[M]. San Mateo, CA, Morgan Kaufmann, 1988

[12] http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/2000/LLS_DDOS_1.0.html