第 36 卷 第 7 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.7 Computer Engineering April 2010
·安全技术· 文章编号：1000—3428(2010)07—0139—03 文献标识码：A 中图分类号：TP393.08

基于动态染色的内存漏洞定位技术

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

摘 要：针对程序漏洞，提出利用基于二进制的程序染色和程序分析技术来检测恶意攻击并有效定位程序漏洞，采用数据依赖关系分析和动态染色的方法，记录起传播作用的写指令及目的内存地址，当检测到漏洞攻击时，通过内存地址找到恶意写指令并定位漏洞。实验结果证明，该方法能成功定位常见内存漏洞的位置，并能定位到有漏洞的库函数的调用点。

关键词：程序漏洞；攻击检测；程序染色；缓冲区溢出；格式化字符串

Memory-related Vulnerabilities Localization Technology Based on Dynamic Tainting

FANG Chen, MAO Bing, XIE Li
(State Key Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

【Abstract】 This paper proposes an efficient mechanism to detect and locate the program vulnerability based on the binary taint analysis and program analysis techniques. The method adopts the data flow analysis and taint analysis. The taint analysis method records the instruction which propagates the taint flag as well as the memory address it writes to. When it detects the attack, it locates the bug by searching the malicious write instruction through the memory address it records. Results of experiments show that the system can localize popular vulnerabilities successfully, and it is able to localize library function call point.

【Key words】 program vulnerability; attack detection; program tainting; buffer overflow; format string

从 1988 年莫里斯蠕虫爆发以来，越来越多的内存相关漏洞在 CERT 上发布出来。程序漏洞的存在已经给社会经济带来了极大的安全隐患。目前很多研究机构都针对这类安全问题提出了检测和防范方法。但是，有关内存漏洞定位方面的工作并不成熟。本文针对内存漏洞的检测和定位提出新的解决方法。

1 背景知识

1.1 内存相关漏洞

内存漏洞是最为常见的程序漏洞，它能够导致攻击者对内存进行任意读写操作，从而控制程序行为，甚至获取系统 root 权限。缓冲区溢出、堆篡改和格式化字符串这 3 类漏洞最为普遍。它们都是由不安全的程序设计语言(如 C)的固有缺陷所导致的。缓冲区溢出[1]是指对缓冲区进行读写时没有“边界检查”而溢出到相邻内存；堆篡改是指堆管理数据被破坏；格式化字符串则是源于对标准输入输出函数不规范的使用。虽然几种漏洞的攻击原理和表现形式各不相同，但都可能导致攻击者对内存进行非法读写操作。

1.2 内存漏洞定位

目前，针对内存漏洞的检测和防范工作已经相对成熟，但是在漏洞自动定位方面的工作还不多见。本文的主要目标是在检测到攻击后能够精确定位漏洞的位置，即非法写内存指令。当非法写指令属于库函数时，则定位到该库函数的调用指令。最后，如果可执行文件中包含 debug 信息，则将指令对应到源代码。

2 系统设计和实现

整个思路依赖于这样一个事实：所有程序受到攻击一定与外部输入相关，即攻击者利用输入数据触发程序漏洞来达到攻击目的。因此，关注的程序都具有输入接口。程序运行时的内存空间可以分成 2 个部分：与外部输入无关和与外部输入相关，利用“染色”[2]技术可以实现对内存的这种区分。当内存与外部数据相关时，则认为其被“污染”了。显然，并不是所有被“污染”的内存就一定是攻击造成的。所以，制定一个合理的检测策略十分重要。本文采用的检测策略与文献[3]中提出的类似，即程序运行时，当发现内存读写或跳转指令的地址操作数与外部输入相关，则认为程序受到了攻击。这样不但能够检测到传统的控制流攻击，而且也能够检测部分数据流攻击。如图 1 所示，攻击检测是异步的，检测到攻击时并不是内存遭到篡改的时刻，而是之后该内存被使用时。因此，采取了从检测点开始，回溯定位漏洞点的方法。下文将介绍解决方法和实现细节。

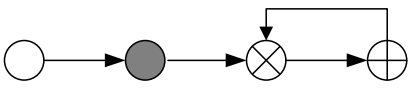

图 1 异步攻击检测和定位

基金项目：国家自然科学基金资助项目(60773171)；国家“863”计划基金资助项目(2007AA01Z448)；江苏省自然科学基金资助项目(BK2007136)

作者简介：房 陈(1982－)，男，硕士研究生，主研方向：软件安全；茅 兵、谢 立，教授、博士生导师

收稿日期：2009-10-24 **E-mail**：leaon_1982@163.com

2.1 系统设计框架

利用程序染色技术跟踪外部输入在程序中的传播情况，在输入数据传播的同时，记录起传播作用的写指令及目的内存地址，当回溯时通过内存地址定位到这条写指令。图 2 就是整个系统的框架图。

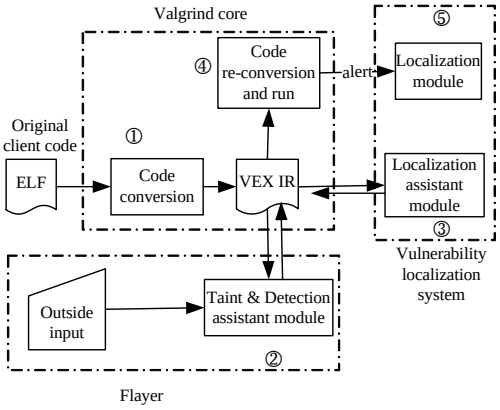

图 2 系统框架图

2.2 系统实现

系统利用工具 Valgrind 和 Flayer 来实现。Valgrind 是一个 x86/Linux 平台下的动态二进制代码分析平台，它将二进制代码转换成 VEX 指令的中间代码。VEX 是一种类 RISC 结构的指令集，访存指令只有 LOAD, STORE 和 JUMP 这 3 类指令。在这 3 类指令前插入检测代码，检查指令的地址操作数是否和外部输入相关。Flayer 是基于 Valgrind 的一种程序分析工具，借助 Flayer 实现程序染色。

图 2 中主要包括有以下几个模块:

①, ④: 二进制代码和 VEX 中间代码的相互转化，由 Valgrind core 自动完成。

②:在 VEX 中间代码中插入记录程序运行时内存状态信息和攻击检测的代码，主要借助 Flayer 工具实现。

③: 以基本块为单位，对块中每条 VEX 中间代码进行分析，找到块内数据之间的依赖关系，并且插入记录(M, W)的代码，其中，W 是与外部数据相关的内存写指令；M 是目的内存地址。

⑤: 检测到非法地址操作数 A 时，通过③中的依赖关系回溯到“污染”A 的内存 M，然后用 M 从(M, W)记录中查到相应的 W，即定位完成。

本文的系统最终可以定位到 W。但是，因为库函数是共享函数，所以库函数内的指令地址是固定的。若非法写指令属于某一个库函数，则系统将定位到该库函数调用点。实现方法是，在函数调用指令 CALL 处检查目的地址是否为 PLT 表中的表项，如果是则为库函数调用，将此调用指令的地址记录下来。当进入库函数体时，传播指令的地址就用此调用指令地址代替并记录下来。当 RET 指令出现时，检查 RET 的目的地址是否为库函数调用点的下一条指令的地址，若是，则刚才调用的库函数返回。

2.3 实例分析

本文举一个简单的例子来说明系统的执行过程，并给出相应的运行结果。图 3(a)是一个存在缓冲区溢出漏洞的程序片段，图 3(b)是 main 函数运行时内存栈的片段。在程序的第 10 行，当输入的字符数大于等于 10 时，就发生了缓冲区溢出，所以第 10 行就是内存篡改点，即漏洞点。溢出可以覆盖 str, I, old ebp 或 ret addr，甚至可以覆盖更高的内存空间。根据前述的检测规则，系统可以在第 23 行检测到 str 指针被外部输入篡改。下文将介绍系统检测攻击和定位漏洞的过程。

(a)缓冲区溢出漏洞实例程序片段　　(b)运行栈片段

图 3 程序实例及其栈结构

图 4 中给出了第 23 行对应的二进制代码及转化后的 VEX 中间代码。其中，LD 和 ST 为内存读写指令，而 GET 是寄存器读指令，tn(n 为数字)表示临时变量。攻击检测代码在 LD, ST 和 JMP 指令前插入，用来检测这些指令的地址操作数是否被外部输入“污染”。在图 4 中，系统在“STle(t4) = 0x78:I8”处检测到异常，因为 t4 所存放的地址值被“染上”了与外部输入相同的颜色。然后通过数据依赖关系分析来回溯定位漏洞点。

```
*(str+i*2) = 'x';
 80484aa:   8b 45 f8    mov    -0x8(%ebp), %eax
 80484ad:   01 c0       add    %eax, %eax
 80484af:   03 45 f4    add    -0xc(%ebp), %eax
 80484b2:   c6 00 78    movb   $0x78, (%eax)
```

⇩

```
->------ IMark(0x80484AA, 3) ------
->t22 = GET:I32(20)
->t21 = Add32(t22,0xFFFFFFF8:I32)
->t23 = LDle:I32(t21)
->------ IMark(0x80484AD, 2) ------
->t1 = Shl32(t23,0x1:I8)
->------ IMark(0x80484AF, 3) ------
->t24 = Add32(t22,0xFFFFFFF4:I32)
->t5 = LDle:I32(t24)
->t4 = Add32(t1,t5)
->------ IMark(0x80484B2, 3) ------
->STle(t4) = 0x78:I8
```

图 4 X86 指令到 VEX 中间代码的转化

图 5 给出了图 4 中 VEX 中间代码片段的数据依赖关系，其中，rn(n 为寄存器的索引值)表示寄存器；G 表示 GET 操作；L 表示 LD 操作，内部字体为斜体的圆圈表示该临时变量的值与外部输入相关。当 t4 被检测到异常时，通过依赖关系图知，t4 依赖于 t1 和 t5，然后检查 t1 的“颜色”，发现 t1 与外部输入无关，接着检查 t5，发现 t5 与外部输入相关，如此继续，直到找到一个与外部输入相关并且其值是从内存 LOAD 进来的临时变量，将此 LOAD 操作的源内存地址记为 M，则 M 就是被篡改的内存地址，再根据 M 从(M, W)记录中找到相应的 W，若 ELF 文件中包含调试信息，则将 W 对应到所在源文件的代码行，定位完成。

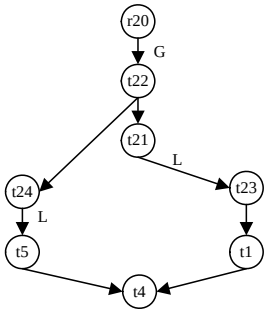

图 5　数据依赖关系

图 6 是系统运行图 3(a)中的程序实例在输入数据“ddddddddddd”后的结果，可以看出系统检测到攻击是在 exp.c 文件的第 23 行，定位到 exp.c 文件的第 10 行。系统可以准确定位到漏洞点，并且能够定位到库函数的调用点。

```
Input: ddddddddddd
==4973== Use of tainted value of size 4
==4973== at 0x80484B2: main (exp.c:23)
==4973== localized at 0x8048479 : main(exp.c:10)
```

图 6　实验结果

3　讨论与后续工作

本文介绍了整个系统的设计思想及实现过程，系统利用了数据依赖关系分析，从检测点回溯定位到漏洞点。实验结果表明，系统可以精确定位漏洞点，并且能够定位到库函数的调用点。目前，大多数的攻击检测技术都是针对控制流数据攻击的，但是最近几年越来越多的非控制流漏洞被发现。系统不仅可以检测和定位控制流攻击，而且可以检测和定位部分的数据流攻击。在已有工作中，文献[4]采用了地址随机化方法检测攻击，所以仅能检测经典的控制流攻击，并且采用了多次重复执行的方法来找到漏洞点，实现复杂而且没有对库函数做特殊的处理。文献[5]采用了指针分析和写集的方法进行实时监测，是一种同步的定位方法，检测到漏洞即定位漏洞。指针分析和写集的使用导致了其检测的不精确，并且实时检测的效率很低。

由于检测策略和文献[3]类似，因此也会出现误报的问题[6]，误报主要是由指针或数组偏移导致的。通过数组和指针访问内存时地址计算方式的不同来区分数组和指针，仅认为指针是不安全的，这就减少了误报。另外，分析数据依赖关系时可能会出现 1 个临时变量依赖 2 个甚至更多被“污染”的临时变量的情况。此时比较临时变量内容的大小，选择内容大的临时变量，因为一般内存地址值都比偏移大，这样做增大了定位的精确度。其次，由于检测策略与文献[3]类似，因此系统不能检测一般意义上的数据流攻击。这也是今后所要重点解决的问题。最后，系统采用染色技术检测攻击，性能有接近 20 倍的损耗。所以，后续的工作主要是进一步降低检测误报率、扩大检测范围以及提高整个系统的性能。

4　结束语

程序漏洞的准确定位对程序补丁、自动攻击过滤以及程序的回退和恢复工作有重要意义，内存漏洞的定位将是笔者未来工作的一个重要研究课题。

参考文献

[1] 苏　朋, 陈性元, 唐慧林. 基于进程执行轮廓的缓冲区溢出攻击效果检测[J]. 计算机工程, 2009, 35(6): 156-158.

[2] Newsome J, Song D. Dynamic Taint Analysis for Automatic Detection, Analysis and Signature Generation of Exploits on Commodity Software[C]//Proc. of the 12th Network and Distributed System Security Symposium. San Diego, USA: [s. n.], 2005.

[3] Chen Shuo, Xu Jun, Nakka N. Defeating Memory Corruption Attacks via Pointer Taintedness Detection[C]//Proc. of IEEE International Conference on Dependable Systems and Networks. Yokohama, Japan: IEEE Computer Society, 2005: 378-387.

[4] Xu Jun, Ning Peng, Kil C. Automatic Diagnosis and Response to Memory Corruption Vulnerabilities[C]//Proc. of the 12th ACM Conference on Computer and Communications Security. Alexandria, USA: ACM Press, 2005: 223-234.

[5] Sezer E C, Ning Peng, Kil C. MemSherlock: An Automated Debugger for Memory Corruption Vulnerabilities[C]//Proc. of the 14th ACM Conference on Computer and Communication Security. Alexandria, USA: ACM Press, 2007: 562-572.

[6] Satoshi K, Hiroyuki K, Ryota S. Base Address Recognition with Data Flow Tracking for Injection Attack Detection[C]//Proc. of the 12th IEEE Pacific Rim Intl. Symposium on Dependable Computing. Riverside, USA: IEEE Computer Society, 2006: 165-172.

编辑　任吉慧

（上接第 138 页）

表 1　协议的比较

协议	效率	安全性	假设
HMQV[5]	2.5E	CK01, KCI, PFS	GDH, KEA1
NAXOS[2]	4E	eCK07	GDH
CMQV[4]	3E	eCK07	GDH
NAXOS+[6]	5E	eCK07	CDH
本文的 AKE-1	4E	eCK07	CDH

5　结束语

本文提出了一个 eCK07 模型安全的基于 CDH 假设和随机预言假设的高效协议 AKE-1，具有较强安全性。另外给出一轮和三轮的变体，不仅保留了 AKE-1 的优良属性，同时满足了一定的应用。下一步研究的目标是设计标准模型(不需要随机预言假设)下的安全高效的协议，以获得更高的安全性。

参考文献

[1] Bellare M, Rogaway P. Entity Authentication and Key Distribution[C]//Proc. of CRYPTO’93. [S. l.]: Springer, 1993: 232.

[2] LaMacchia K, Lauter K, Mityagin A. Stronger Security of Authenticated Key Exchange[C]//Proc. of ProvSec’07. Heidelberg, Germany: Springer, 2007: 1-16.

[3] Cash D, Kiltz E, Shoup V. The Twin Diffie-Hellman Problem and Applications[C]//Proc. of EUROCRYPT’08. Heidelberg, Germany: Springer, 2008: 127-145.

[4] Ustaoglu B. Obtaining a Secure and Efficient Key Agreement Protocol for (H)MQV and NAXOS[J]. Designs, Codes and Cryptography, 2008, 46(3): 329-342.

[5] Krawczyk H. HMQV: A High-performance Secure Diffie-Hellman Protocol[C]//Proc. of CRYPTO’05. Heidelberg, Germany: Springer, 2005: 546-566.

[6] Lee J, Park J. Authenticated Key Exchange Secure Under the Computational Diffie-Hellman Assumption[EB/OL]. (2009-04-08). http://eprint.iacr.org/2008/344.

编辑　任吉慧