缓冲区溢出漏洞研究与进展*)

李毅超 刘 丹 韩 宏 卢显良
(电子科技大学计算机科学与工程学院 成都 610054)

摘 要 软件中的缓冲区溢出漏洞是个严重的安全隐患,利用它的攻击给社会造成了巨大的危害和经济损失,缓冲区溢出漏洞检测防护技术已成为一个研究热点。首先分类剖析了缓冲区溢出攻击原理,进而对缓冲区溢出漏洞检测防护技术十几年来的研究进展进行了讨论,最后给出该领域的研究热点问题与展望。

关键词 缓冲区溢出,静态分析,动态分析,边界检查

Research and Development of Buffer Overflow Vulnerability

LI Yi-Chao LIU Dan HAN Hong LU Xian-Liang
(School of Computer Science & Engineering, UEST of China, Chengdu 610054)

Abstract The buffer overflow vulnerability in software results in serious security problem, and attacks based on buffer overflow have brought huge society chaos and economic loss. Many researchers focus on the detection and defense of buffer overflow. In this paper, the execution mechanism of the buffer overflow attacks is first presented, then the progress in detection and defense of buffer overflow is addressed, and finally some major problems and research trends in the area are discussed.

Keywords Buffer overflow, Static analysis, Dynamic analysis, Bound checking

从1988年蠕虫病毒至今,缓冲区溢出作为一种安全危害级别相当高的计算机安全问题而一直存在。据来自CERT的报告:2003年,缓冲区溢出漏洞占严重漏洞的70.4%,该比例随着时间还在上升。缓冲区溢出攻击已成为安全威胁中最为严重的一类。

本文第1节阐述了缓冲区溢出攻击原理。第2节分类介绍和评价了当前主流的缓冲区检测防护技术。最后对缓冲区检测防护技术的未来发展趋势进行了分析和预测。

1 缓冲区溢出攻击的原理

缓冲区溢出与程序在内存中的组织分布有关,在冯·诺依曼计算机体系结构中程序的数据和执行代码混合存储,对数据的修改可造成程序代码的修改。其直接原因是C语言本身缺乏类型安全,追求性能而忽视正确性,不检查边界直接对内存进行操作。利用缓冲区溢出漏洞的攻击原理主要有以下四种情形。

1.1 堆栈溢出攻击

图1中,当调用函数copy()后,参数、返回地址、前一个栈的指针、本地变量被依次压入栈中。图中箭头指明堆栈和内存的增长方向。函数strcpy()将输入复制到buffer()。由于strcpy()不检查输入变量的边界,当它向buffer()复制的字符多于512个,输入字符就可覆盖堆栈中的返回地址并使其指向攻击代码。当程序从copy()返回到main()时,攻击代码被执行。不过,目前几乎所有的解决方案都能检测和阻止这类堆栈攻击。

堆栈攻击的一个变种是,只覆盖前一个栈的栈指针,在从copy()返回时,这个栈指针会指向main(),当程序从main()返回时,攻击被激活。这类攻击能突破StackGuard[1]和StackShield[2]。

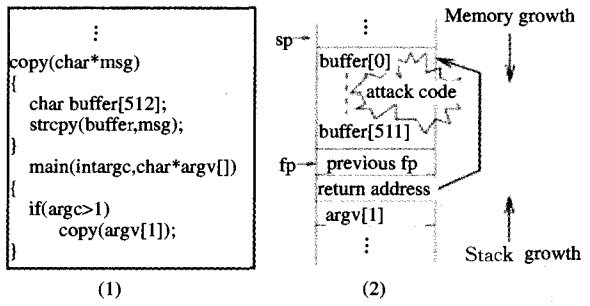


图1 堆栈溢出攻击

1.2 本地指针溢出攻击

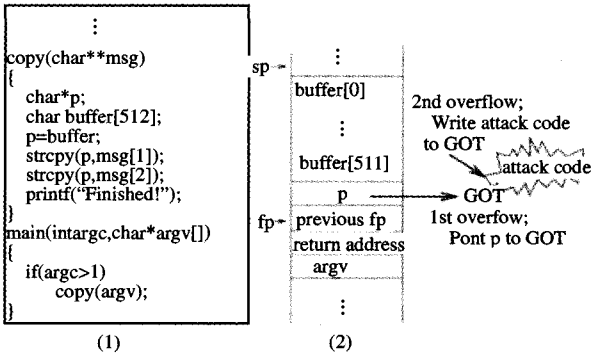


图2 本地指针溢出攻击

*)科技部国家科技基础条件平台专项(2003DIA7J051)。**李毅超** 硕士,副教授,主要研究领域为计算机网络及网络安全;**刘 丹** 博士,副教授,主要研究领域为分布式系统及网络安全;**韩 宏** 博士,副教授,主要研究领域为操作系统及网络安全;**卢显良** 教授,博士生导师,主要研究领域为操作系统及网络安全。

即使返回地址和栈指针受到了保护，缓冲区溢出仍可能发生。图2中，函数copy()有个本地指针p，并两次调用函数strcpy()。在堆栈中，指针p位于buffer[]下面。攻击过程是：

1)在执行第一个strcpy()时，攻击代码被注入到缓冲区中，p被覆盖并指向函数printf在全局偏移表GOT(Global Offset Table)中的入口；

2)通过第二个strcpy()函数将攻击代码的地址复制到printf在全局偏移表GOT中的入口中；

3)当程序执行copy()函数中的printf("Finished!")时，激活攻击代码。

该例说明GOT表也是引发溢出攻击的部位，应得到保护。

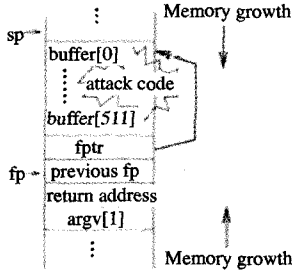
1.3 本地函数指针溢出攻击

本地函数指针也可以被利用成为溢出攻击。图3(1)，在copy()函数中，strcpy()函数首先被调用，然后fptr所指向的函数被执行。图3(2)，在堆栈中，fptr位于buffer[]下，这样，fptr就可能被覆盖并指向攻击代码。执行(void)(*fptr)(buffer)时，就会激活攻击代码。

```
int good(const char*str){
.....
}
copy(char*msg){
int(*fptr)(const char*str);
char buffer[512];

fptr=(int(*)(const char*str))good;
strcpy(buffer,msg);
(void)(*fptr)(buffer);
}
main(intargc,char*argv[])
{
 if(argc>1)
     copy(argv[1]);
}
```

(1)



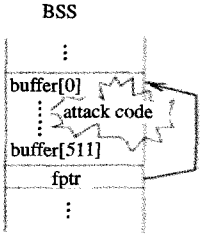
(2)

图3 本地函数指针溢出攻击

1.4 BSS函数指针溢出攻击

```
static char buffer[512];
static int (*fptr)(const char*str);
int good(const char*str){
     .....
}
copy(char*msg){
fptr=(int(*)(const char*str))good;
strcpy(buffer,msg);
(void)(*fptr)(buffer);
}
main(intargc,char*argv[])
{
if(argc>1)
copy(argv[1]);
}
```

(1)



(2)

图4 BSS函数指针溢出攻击

图4中，对BSS(Block Storage Segment)中函数指针的利用。由于利用发生在BSS中，诸如StackGuard、IBM SSP等这样的堆栈溢出防护技术对这类攻击无效。

2 检测与防护研究进展

2.1 基于目标代码的分析技术

文[1]介绍的Fuzz工具，用于对目标代码进行完全的黑箱式分析。Fuzz执行时间很难估计，而且该工具给出的仅仅是一种定性判断，漏洞在源码中的具体位置不能确定，而且没有后续研究。

2.2 基于源码的分析方法

2.2.1 基于源码的静态检测方法

David Evans[3]提出一种使用规范来检查代码安全性的方法，并开发了LCLint工具。LCLint使用C源代码文件和一系列的LCL语言编写的规范文件作为输入，然后自动检查源文件和规范文件及其编程传统之间的不一致性，输出相应的警告报告。同一般的程序分析工具相比，LCLint可以检查抽象边界问题，全局变量的非法使用问题等，因而可以作为源代码缓冲区漏洞检测的基础之一。2001年David Larochelle[4]等在LCLint的基础上，提出了一种使用约束分析技术来解决漏洞检测问题的方案。其性能等价于一般的编译器性能。

Yichen Xie[5]开发了一种名为ARCHER的检查工具，其研发背景是斯坦福网络研究中心和DARPA之间的合作项目。其首先对源码进行语法分析，生成抽象语法树，然后利用工具构造相应的程序流程控制图，在其上进行路径敏感分析。这类基于抽象语法树的检测方法的变种很多[6~8]。

这类基于源代码的静态分析技术研究比较活跃。其目标是在软件开发过程中阻止溢出，具有应用前景和应用市场。缺点是不适用于不具备源码的遗产软件或第三方提供的二进制软件。

2.2.2 基于源码的动态检测方法

典型的研究[9]是通过一些反汇编工具对目标代码进行处理，然后再依赖一些源代码静态检测技术进行处理。Eric Haugh和Matt Bishop[10]提出了一种动态检测的方法。通过在源代码文件中引入一系列的assert之类的语句，跟踪内存中的缓冲区，从而在程序运行期间加以判断。Ghosh[11]提出的故障注入法，则是将攻击代码插入被测软件以测试是否存在漏洞。动态检测方法研究不活跃，但故障注入法有着广泛的应用，是目前软件漏洞发掘的常用方法之一。

2.3 基于编译器的方法

StackGuard和StackShield都是GNU C编译器的扩展工具，用于检测调用的返回地址是否正常。StackGuard是紧接返回地址存储一个canary随机值，同时在通用寄存器中保留一份canary，程序执行时检查canary的完整性，不足之处是canary值可被攻击[12]。类似的还有其它一些编译器扩展方法和工具[13~15]。PointGuard[16]是StackGuard的一般化方法，在函数指针、longjmp缓冲区之后增加canary保护函数指针，防止被缓冲溢出操作所改写，并与编译器整合优化。该方法结合StackGuard可有效地防止缓冲区溢出。对于少数溢出其它变量的攻击，通过强制对某一变量加入附加字符来实现保护。Kyung-suk Lhee[17]提出一种针对C库函数调用产生的溢出问题的动态检测技术。其对编译器进行扩展，使得编译后的程序在内存中维持有局部变量、函数参数、全局变量等类型的缓冲区类型信息，并对内存堆中的缓冲区大小进行记录；依赖这些信息，可对某些溢出情形作出判断。Rinard. M等[18]开发了编译器工具，动态检测存储器的越界访问。Zhu. G等[19]提出了重定位函数指针方案，当函数指针存入内存时加密，读取时解密。

StackGuard的出现，具有里程碑的作用，开创了在编译器解决溢出的先河，至今基于编译器的研究仍是主流方向之一。

2.4 基于硬件的方法

M. Frantzen等[20]为SPARC系统设计了一个方案

StackGhost,设置寄存器窗口用于存放返回地址,操作系统负责窗口数据的填充和提取,从而阻止对返回地址的修改。Shaolizi[21]提出了基于处理器的边界检查方案,Madan[22]提出在堆栈中放入返回地址的同时,也装入该地址的加密值。当返回指令访问返回地址时,先将返回地址加密后与其存储的加密值作一比较,以判断是否一致,从而判断是否存在溢出漏洞,并进行阻断。

Ruby B. Lee[23]等人对处理器进行改进,引入一个安全返回地址栈(SRAS),在程序执行时保存其正确的返回地址。只要调用和返回指令可修改 SRAS 的值。处理器在执行返回指令时采用的是 SRAS 中的返回地址。如果 SRAS 中的返回地址和内存中的返回地址不一致,说明内存中的返回地址可能受到攻击。

近几年,基于硬件的研究方法主要是对处理器进行改造,并已推出产品,如 AMD Athlon-64 芯片就可阻止缓冲区溢出攻击[24]。自从 AMD Athlon-64 芯片的诞生,以及 Intel 的跟进研究,使得基于硬件的方法成为目前及今后的研究热点。该方法的优势是无需重新编写和编译源代码,对于软件开发者和软件使用者都适用。

2.5 改造 C 语言库函数

A. Baratloo 等人研发了 libsafe 工具[25]和 libverify[26]工具。前者解释不安全的 C 语言库函数,进行边界检查,从而保护帧指针和返回地址;不足之处是只能阻止源于某些 C 语言库的缓冲区溢出。后者在堆中保存每个函数和每个返回地址。不足之处是应用程序降低 15% 性能,代码空间增大 2 倍,不适合内存小的嵌入式设备。与边界检查相比,指针完整性检查并不能防止所有的缓冲区溢出问题,但在性能和兼容性上具有相当的优势;边界检查对每次数组的操作都进行检查,程序指针检查只在被引用时进行检查,无论在 C 还是在 C++ 中,后者的引用频率都要小得多。目前边界检查和指针完整性检查趋于在编译层和硬件层完成。

2.6 基于操作系统的方法

Nebenzahl 等[27]提出一种针对 windows 操作系统的堆栈防御技术,开发了一个 windows 的溢出免疫 DLL。Shaozili[28]提出一种基于操作系统内核的内存边界保护方法。基于操作系统的解决方案是目前正在形成的研究热点。操作系统比硬件和编译器掌握了更多关于运行程序的信息,能够更有效地解决计算机安全问题。例如,边界检查或边界完整性检验都可由内核通过存储边界值来完成。基于操作系统的机制常常和基于硬件的机制结合使用,AMD Athlon-64 芯片和 windows 系统就是一个很好的例子。

2.7 其他防护技术

通过不安全的 C 语言进行改造,从而阻止溢出漏洞的产生。如 L. Hornof 和 T. Jim[29]对 C 语言进行改进,提出了 Cyclone 安全编程语言。该方法不足之处是,程序员须清楚其与 C 语言的差异。

P. A. Karger[30]等人提出将内存栈存放在非执行页面中的防护技术;Yu-an Tan[31]等人提出不可执行堆栈方法,性能优于基于页面的不可执行堆栈方法。该类技术的不足是:难以兼容现有的应用程序;另外,某些操作系统不支持非执行堆栈。

3 发展趋势

缓冲区溢出检测及防护技术的研究总趋势是:从软的层面走向硬的层面;从单机制走向多机制的结合。

AMD Athlon-64 芯片的成功推出,以及 Intel 芯片的积极跟进,使研究者们认识到,芯片级解决方案的应用前景。目前,单依赖芯片也难以彻底解决缓冲区溢出问题,例如,AMD Athlon-64 芯片只能解决堆栈溢出问题,对指针溢出无能为力。多种检测防护机制的融合,是下一步的一个重要研究方向。如何在程序开发阶段消除溢出漏洞,仍是一个重要的问题。源码静态分析技术,仍会是研究热点。今后的工作集中在性能及实用性的提高上。

基于操作系统的解决方案会成为下一步研究热点。计算机信息安全问题不仅仅是溢出问题,还有诸如恶意代码、安全审计等,而这些问题都可以由操作系统一体化解决,而编译层和硬件层都不具备这个优势。

另外,如何恢复发生溢出的程序,会成为下一步受关注的问题。目前,对发生溢出的程序采用的方法是关闭,这意味着程序溢出前的工作都浪费了,严重降低了资源的效率。如何使发生溢出的程序继续工作,有待进一步研究。

参 考 文 献

1 Cowan C, et al. StackGuard: Automatic Detection and Prevention of Buffer-overrun Attacks. In: Proceedings of the 7th USENIX Security Symposium, January 1998
2 StackShield. http://www.angelfire.com/sk/stackshield
3 Evans D, Guttag J, Horning J, Tan Y M. LCLint: A Tool for Using Specifications to Check Code. In: SIGSOFT Symposium on the Foundations of Software Engineering, December 1994
4 Larochelle D, Evans D. Statically detecting likely buffer overflow vulnerabilities. In: USENIX Security Symposium, Washington, D. C., August 2001
5 Xie Yichen, Chou A, Engler D. ARCHER: Using Symbolic, Pathsensitive Analysis to Detect Memory Access Errors. ESES/FSE'03, Helsinki, Finland, September 2003
6 Dor N, Rodeh M, Sagiv S. CSSV: towards a realistic tool for statically detecting all buffer overflows in C. PLDI, 2003. 155～167
7 Wagner D. Static Analysis and Computer Security: New Technique for Software Assurance: [PHD Dissertation]. Fall 2000
8 Ganapathy V, Jha S, Chandler D, et al. Buffer Overrun Detection using Linear Programming and Static Analysis. CCS'03, Washington, DC, USA, October 2003
9 Gillette T B. A Unique Examination of the Buffer Overflow Condition. Bachelor of Sience Ocean Engineering, Florida Institute of Technology, May 2002
10 Haugh E, Bishop M. Testing C Programs for Buffer Overflow Vulnerabilities. In: the 10th Annual Network and Distributed System. Security Symposium Catamaran Resort Hotel San Diego, California, February 2003
11 Ghosh A, O'Connor T. Analyzing Programs for Vulnerability to Buffer Overrun Attacks: [Technical Report]. Reliable Software Technologies, January 1998
12 Bulba, Kil3r. Bypassing StackGuard and StackShield. Phrack Magazine, 2000, 10(56)
13 Baratloo A, Singh N, Tsai T. Tranaparent runtime defense against stack smashing attacks. In: Proceedings of the 2000 USENIX Annual Technical Conference, San Jose, CA, June USENIX, 2000. 251～262
14 PaX. http://pageexec, vertualave. net
15 SolarDesigner, Non-executable stack patch. http://www. openwall. com/linux
16 Cowan C, Beattie S, Johansen J, Wagle P. Pointguard: Protecting Pointers from Buffer-Overflow Vulnerabilities. In: Proc. USENIX Security Symp., Aug. 2003
17 Lhee K S, Chapin S J. Type-Assisted Dynamic Buffer Overflow Detection. In: USENIX Security Symposium, 2002. 81～88
18 Rinard M, Cadar C, Dumitran D, Roy D M, Leu T. A dynamic technique for eliminating buffer overflow vulnerabilities (and other memory errors). In: Computer Security Applications Conference, 20th Annual, 2004
19 Zhu G, Tyagi A. Protection against indirect overflow attacks on pointers. In: Information Assurance Workshop, Proceedings. Second IEEE International 2004, 2004. 97～106

(下转第 125 页)

显。如何对待选中间视图预先进行过滤以缩减考察空间，将作为我们进一步的研究方向。此外，算法对非均衡数据源的更新处理也有待进一步研究。

参 考 文 献

1 Gupta A, Mumick I. Maintenance of materialized views: problems, techniques, and applications. IEEE Data Eng Bull, 1995, 18(2): 3～19

2 陈长清，程恳，冯玉才. 一种用于搜索可能响应查询的候选实化视图的索引. 计算机科学，2003，30(5)：176～179

3 Salem K, Beyer K S, et al. How to roll a join: asynchronous incremental view maintenance. In: Proceedings of SIGMOD, 2000. 129～140

4 Lee K Y, Son J H, et al. Efficient incremental view maintenance in data warehouse. In: Proceeding of CIKM, 2001. 249～356

5 Liu B, Rundensteiner E A, Finkel D. Maintaining large update batches by restructuring and grouping. Information Systems, 2007 (Accepted)

6 Lee K Y, Kim M H. Optimizing the incremental maintenance of multiple join views. In: the Proceeding of DOLAP, 2005. 107～113

7 王新军，洪晓光，等. 数据仓库中多数据源物化视图的一种有效更新算法. 计算机研究与发展，2004，41(5)：874～879

8 Labio W J, Yerneni R, et al. Shrinking the warehouse update window. In: Proceedings of SIGMOD. June 1999. 383～395

9 DeHaan D, Larson P-A, et al. Stacked indexed views in Microsoft SQL server. In: Proc of SIGMOD, 2005. 179～190

10 Folkert N, Gupta A, et al. Optimizing refresh of a set of materialized views. In: the Proceeding of 31st VLDB, 2005. 1043～1054

11 Skowron A, Rauszer C. The discernibility matrics and function in information system. In: Slowinski R, ed. Intelligent Decision Support Handbook of Application and Advances of the Rough Sets Theory. Dordreecht: Kluwer Academic Publishers, 1991. 331～362

12 TPC BENCHMARKTM H Standard Specification (Decision Support), Revision 2.3.0. http://www.tpc.org/tpch

(上接第 89 页)

20 Frantzen M, Shuey M. StackGhost: Hardware Facilitated Stack Protection. In: Proceedings of the 10 th USENIX Security Symposium, August 2001

21 Shao Z, Cue Chun, Zhuge Qingfeng, H M Sha E. Security Protection and Checking in Embedded System Integration Against Buffer Overflow Attacks. In: Proceeding of the International Conference on Information Technology: Coding and Computing, 2004

22 StackOFFence: a technique for defending against buffer overflow attacks. Madan, B B, Phoha S, Trivedi K S, et al. Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on Volume 1, April 2005. 656～661

23 Lee R B, Karig D K, McGregor J P, et al. enlisting hardware architecture to thwart malicious code injection. Lecture Notes in Computer Science, 2004, 2802: 237～252

24 Paulson L D. 同 New chips stop buffer overflow attacks. Computer, 2004, 37(10): 28

25 Baratloo A, Singh N, Tsai T. Transparent Run-time Defense against Stack Smashing Attacks. In: Proc. of the 9th USENIX Security Symposium, June 2000

26 Baratloo A, Singh N, Tsai T. Transparent Run-Time Defense Against Stack SmashingAttacks. In: Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, California, USA, June 2000

27 Nebenzahl D, Sagiv M, Wool A. Install-Time Vaccination of Windows Executables to Defend against Stack Smashing Attacks. Dependable and Secure Computing, IEEE Transactions on, 2006, 3(1): 78～90

28 Shao zili, Xue Chun, et al. Efficient array and pointer bound checking against buffer overflow attacks via hardware/software. In: International Conference on Information Technology: Coding and Computing, ITCC, v1. Proceedings ITCC 2005 - Intenational Conference on Information Technology: Coding and Computing, 2005. 780～785

29 Hornof L, Jim T. Certifying Compilation and Run-time Code Generation. In: Proceedings of the ACM Conference on Partial Evaluation and Semantics-Based Program Manipulation, January 1999

30 Karger P A, Schell R R. Thirty Years Later: Lessons from the Multics Security Evaluation. In: Proceedings of the 2002 Annual Computer Security Applications Conference, December 2002. 119～126

31 Tan Yu-an Tan, Zheng Ji-yan Zheng, Cao Yuan-da, Zhang Xue-lan. Buffer overflow protection based on adjusting code segment limit. Communications and Information Technology, 2005. ISCIT 2005. In: IEEE International Symposium on Volume 2, 2005. 947～950

(上接第 102 页)

载体图像(a)的大小为 256×256，如果 α=1，修改一位只嵌入一比特位信息，则可嵌入的最大信息量为 256×256 / 8 =8192byte，实验中实际嵌入的秘密信息图(b)大小为 90×90 =8100byte，若 α=2，则可在对载体图像修改相同位数的条件下，增加一倍的嵌入量，从实验结果可看出，单靠人眼视觉很难发现载体图像中藏有秘密信息，两种算法的隐蔽性均较好，并且从上面的分析也可知本文算法对常规的密写分析方法有较强的抵抗能力，算法是有效的、可行的。

4 分析与讨论

LSB 信息隐藏是一种基于空域的隐藏算法，空域隐藏算法采用直接改变图像元素值的方法，一般是在图像元素的亮度或色度中加入隐藏的内容，空域类算法的特点是只需对隐秘载体进行很小的、不易察觉的改变就能隐藏很大的信息量，并且信息嵌入和提取算法简单，待隐藏信息格式又可以是任意的文件，因此计算速度较快。同时人眼对绿色最敏感，红色次之，蓝色最不敏感。因此对于载体图像为 24 位位图的图像，可在人眼不敏感的颜色位中嵌入较多比特位，而在敏感的颜色位中嵌入较少比特位，在不影响视觉感观的条件下，增加嵌入量。但其存在的主要局限是对信道干扰及数据操作的抵抗力较弱，当载体文件遭到破坏或受到某些程度噪声的干扰时，嵌入的隐藏数据将很难完整地提取出来。不过，作为一种大数据量的信息隐藏方法，LSB 算法在隐藏通信中仍占据着重要的地位，他也是其它隐藏方法的基础。

另外，目前已有很多变换域隐藏算法，变换域隐藏算法是利用某种数学变换，将图像用变换域(如频域)表示，通过更改图像的某些变换域系数加入待隐藏信息，然后再利用反变换来生成原始图像。常见的变换域算法有：基于 DCT 的变换域算法、基于 DWT 的变换域算法等，这些都是研究的热点。

参 考 文 献

1 Aucsmith D. Information hiding. In: Proceedings of the Second International Workshop, Lecture Notes in Computer Science 1525. Berlin: Springer-V erlag, 1998

2 Westfeld A, Pfitzmann A. Attacks on steganographic systems. In: Pfitzmann A, ed. Proc. of the 3rd Int'l. Workshop on Information Hiding. Lecture Notes on Computer Science 1768, Berlin: Springer-Verlag, 1999. 61～76

3 Chang C C, Lin M H, Hu Y C. A Fast and Secure Image Hiding Scheme Based on LSB Substitution [J]. International Journal of Pattern Recognition and Artificial Intelligence, 2002, 16(4): 399～416

4 王道顺，齐东旭. 一种新的数字图像隐藏方案[J]. 计算机学报，2000，23(9)：949～952

5 刘年生，郭东辉. 基于混沌加密的一种图像信息隐藏传送方法[J]. 计算机工程，2006，32(7)：135～137

6 王君，田玉敏，李春霞. 一种改进的图像自适应信息隐藏算法[J]. 计算机应用研究，2005，5(3)：145～147

7 吕欣，马智，冯登国. 安全隐写系统的信息理论分析[J]. 计算机科学，2006，33(6)：140～142