缓冲区溢出攻击模式及其防御的研究

程红蓉，秦志光，万明成，邓 蔚

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】借助统一建模语言，概括近十年来利用缓冲区溢出进行攻击的攻击模式，从预防、发现、抵御缓冲区溢出攻击以及攻击后的程序恢复等方面对目前有代表性的防御、检测方法和攻击恢复技术进行了归纳、分析和比较，指出这些方法和技术的弊端以及可能采取的规避手段。提出了在攻击技术不断发展的情况下，彻底、有效地解决缓冲区溢出所面临的问题，编写安全的程序是解决缓冲区溢出的关键，并对将来解决缓冲区溢出可采用的有效方法和手段进行了讨论。

关 键 词 攻击模式； 缓冲区溢出； 防御方法； 统一建模语言； 安全编程
中图分类号 TP311 文献标识码 A

On the Buffer Overflow Attack Mode and Countermeasures

CHENG Hong-rong，QIN Zhi-guang，WAN Ming-cheng，DENG Wei

(School of Computer Science and Engineering, University of Electronic Science and Technology of China Chengdu 610054)

Abstract A general model of buffer overflow based attacks is described by unified modeling language. The analysis and comparison of the existing representative methods and apparatuses of defense and recovery against buffer overflow attacks are presented, including analyzing their vulnerabilities and possible means to bypass them. Highlighting the state-of-art challenging issues for facing the tradeoff of security and performance efficiency, and the continuing evolution of attach techniques, it is pointed out that security programming is the key to solve buffer overflow problems. Finally, some technical trends are given.

Key words attack model; buffer overflow; countermeasures; unified modeling language; security programming

1 缓冲区溢出的攻击模式

缓冲区定义为连续的一段存储空间。当写入缓冲区的数据量超过该缓冲区能容纳的最大限度时，缓冲区溢出发生，溢出的数据将改写相邻存储单元上的数据。造成缓冲区溢出的根本原因是代码在操作缓冲区时，没有有效地对缓冲区边界进行检查。缓冲区溢出可以成为攻击者实现攻击目标的手段，但是单纯地溢出缓冲区并不能达到攻击的目的。在绝大多数情况下，一旦程序中发生缓冲区溢出，系统会立即中止程序，并报告“段错误”。只有对溢出缓冲区适当地加以利用，攻击者才能通过其实现攻击目标。

根据实现目标的不同，缓冲区溢出攻击分为和改变程序逻辑两类攻击。目前已出现的利用缓冲区溢出进行的攻击主要属于改变程序逻辑。与破坏敏感数据的攻击相比，此类攻击的目标不是仅仅针对某个或者某些数据，而是针对整个被攻击系统；虽然破坏的不是敏感数据，但是攻击者可以通过改来变这些数据来改变原有的程序逻辑，以此获取对本地/远程被攻击系统的控制权。

实施缓冲区溢出攻击的具体方法多种多样，其攻击手段为如图1所示的同一种模式。图中，缓冲区溢出的攻击模式用统一建模语言(UML)的泳道(Swimlanes)活动图进行描述。对于仅破坏敏感数据的缓冲区溢出攻击，只涉及该模式的前两个活动(Activity)，可以概括为以下步骤：

1) 注入恶意数据

恶意数据是指用于实现攻击的数据，它的内容将影响攻击模式中后续活动能否顺利进行。恶意数据可以通过命令行参数、环境变量、输入文件或者网络数据注入被攻击系统。

2) 溢出缓冲区

制造缓冲区溢出的前提条件是发现系统潜在

收稿时间：2007– 08 – 25

作者简介：程红蓉(1975 –)女，博士生，讲师，主要从事信息安全、模式识别及其应用等方面的研究；秦志光(1956 –)，男，博士，教授，博士生导师，主要从事信息安全、模式识别等方面的研究；万明成(1985 –)，硕士生，主要从事模式识别方面的研究；邓 蔚(1979 –)，男，博士生，主要从事信息安全、数据挖掘及其应用等方面的研究.

的、可被利用的缓冲区溢出隐患。可被利用是指该隐患在特定的外部输入条件作用下，可导致缓冲区溢出的发生，图1的缓冲区溢出攻击模式即是建立在该前提条件成立的基础上。C/C++是最常见的不安全编程语言，用其编写的程序容易存在缓冲区溢出。

3) 控制流重定向

控制流重定向是将系统从正常的控制流程转向非正常控制流程的途径，传统做法通过改写位于堆栈上的函数返回地址来改变指令流程[1]，并借助“NOP”指令提高重定向的成功率[2]。除此之外，其他可用于控制流重定向的方法有：(1) 改写被调函数栈上保存的调用函数栈的栈基址。(2) 改写指针。如在Linux系统中，借助ELF文件格式的特性改写过程链接表(PLT)、析构函数段(.dtor)、全局偏移表(.got)中的函数指针，可以改变控制流程[3]。此外，通过对数据指针、虚函数指针(VPTR)、异常处理指针的改写也可以改变控制流程[4]。(3) 改写跳转地址，当跳转指令执行时实现控制流重定向。

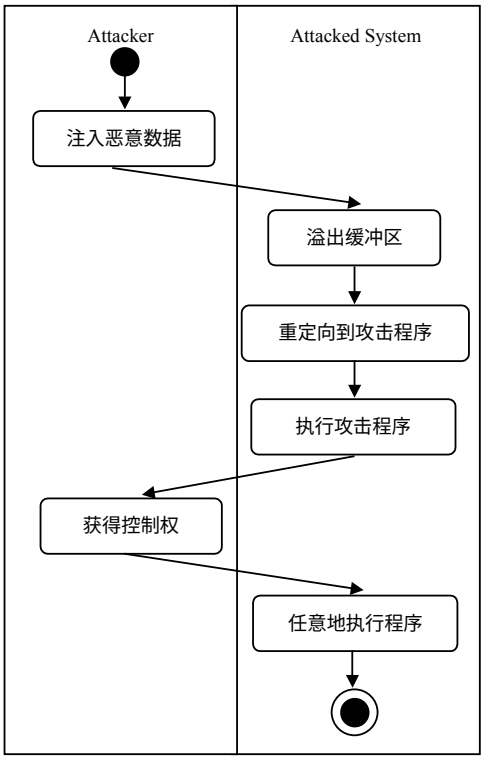

图1 缓冲区溢出攻击模式

4) 执行攻击程序

当控制流被成功地重定向到攻击程序所在位置时，攻击程序得以运行，实现攻击目标。攻击程序专指真正实现攻击的代码部分，称为有效载荷(Payload)。在攻击中，有效载荷可能以可执行的二进制代码形式放置在恶意数据中，这种有效载荷用于产生命令解释器-shell[3]，称为Shellcode。此外，有效载荷也可能是已经存在于内存中的代码，相应的攻击技术称为Arc Injection[4]。

2 缓冲区溢出问题的预防

对缓冲区溢出攻击的防御包括预防缓冲区溢出的发生和抵御利用缓冲区溢出进行的攻击。两者在对抗缓冲区溢出攻击方面互为补充，形成多层次的防御体系，但是两者的出发点存在本质不同。“预防”是从根本上避免缓冲区溢出问题的出现。

2.1 类型安全的编程语言

适当地应用Java、C#、Lisp、ML等类型安全的编程语言可以有效地减少使用C/C++时的安全风险。然而即使是类型安全的编程语言也无法提供绝对的安全[5]，而且安全性与时间效率始终是对立量。在提供了安全的同时，安全编程语言也损失了执行效率。

2.2 增强C/C++的类型安全

C/C++语言的灵活性导致了类型的不安全。为了加强语言安全，出现了以下对C/C++编译器或者C/C++语言本身进行改写的方法：(1) 文献[6]修改了原有的GCC编译器，加入对指针和数组的边界检查。测试表明，修改后的编译器使大多数程序的执行效率降低了5～6倍。(2) 文献[7]在保留C的语法和语义的基础上，加强了语言限制(如禁止指针运算、不支持setjmp/longjmp调用)，开发出安全的C语言-Cyclone。增加的语言限制降低了C语言的执行效率，同时也引起兼容性问题。

2.3 安全编程

在程序开发阶段中，安全编程思想对引导编程人员如何避免编写存在缓冲区溢出等安全隐患的程序具有重要作用。Michael针对如何编写安全的C/C++程序提出若干建议和方法。实施安全编程需要对编程人员进行相关培训，但是目前有关安全编程的培训并不普及，而且即使是受过培训的编程人员也很难一直都能编写无安全隐患的程序，因此有必要借助静态分析和动态检测，对程序进行安全方面的检验。

2.4 静态分析

静态分析是检测软件错误的重要方法。近几年，静态分析逐渐扩展到安全领域，应用于安全问题的检测。目前针对缓冲区溢出问题的检测，有代表性的静态分析技术主要有：(1) 简单的字符匹配查找。Unix/Linux的grep命令可以作为区溢出最简单的静

态分析工具。(2) 词法分析(Lexical Analysis)。比Grep进一步，词法分析工具如ITS4[8]、RATS和Flawfinder采用模式匹配进行静态分析。这种方法的显著优点是分析效率高，但是由于词法分析不考虑程序的语义，会产生大量的虚假警报(False Alarm)，严重影响了工具的有效性。(3) 整数范围分析。文献[9]将缓冲区形式化为表示缓冲区大小和使用大小的一对整数，将边界检测转化为整数计算，开发出检测工具BOON。尽管BOON发现了许多使用词法分析技术未能检测出的缓冲区溢出问题，由于不考虑程序的语义，同样会产生大量的虚假警报。文献[10]基于线性规划(LP)的算法进行整数范围分析，加入部分语义分析，取得了比BOON更好的效果。(4) 基于程序注解的分析。文献[11]给出基于程序注解的轻量级(Light-weight)静态分析工具LCLint及其后续版本Splint[12]，可用于大型程序的检测。与Splint不同，文献[13]给出的重量级(Heavy-weight)静态分析工具CSSV旨在检测出所有缓冲区溢出。基于程序注解的静态分析方法实现了一定程度的语义分析，但是这种方法的自动化程度不够理想，并且注解的准确度和详细度都将直接影响分析效果。(5) 基于调用图/控制流图的分析。文献[14]给出的静态分析工具ARCHER将源程序转化为抽象语法树(AST)，进而构建出调用图进行分析。文献[15]给出的基于编译器扩展的静态分析工具UNO对由抽象语法树转换的控制流图(CFG)进行分析，并且UNO支持对用户自定义属性的检测。基于调用图/控制流图的分析方法可以有效地模拟出程序的数据/控制流程，然而这些工具都只局限于对部分问题的分析，因而只能检测出部分隐患。文献[16]详细列举出ARCHER和UNO未作分析的方面。(6) 基于内存状态的分析。静态分析工具通常需要具有被分析程序的源代码，但是有时无法获得源代码。文献[17]提出一种静态分析技术，将内存模型化为若干状态，结合程序注解对外来不可信机器码(SPARC机器码)进行分析。

ITS4 RATS、FlawFinder、BOON、CSSV、Splint、ARCHER和UNO的对比如表1所示。

一个理想的静态分析工具应该有100%的检测率和0%的虚假警报率(False Alarm Rate)，并且具有检测效率高、易用程度高(包括自动化程度)、对兼容性的影响低和开发成本低的特点。由静态分析技术的现状可知，理想和现实还有很大的差距：(1) 检测率较高的工具检测效率低，无法应用于大型程序的检测。(2) 虚假警报率低的工具通常以牺牲检测率为代价。(3) 检测率较高、虚假警报率低的工具由于复杂度高，检测效率和易用程度都受影响，而且需要较高的开发成本。有关部分静态分析工具的评估可参考文献[18-19]。

表1　静态分析工具对比

工具名	可用性	分析策略	语义分析程度		分析范围	
			流有关	上下文有关	过程间	过程内
ITS4 RATS FlawFinder	开放源码	词法分析				
BOON	开放源码	整数范围分析			√	
CSSV	研究	基于程序注解		√	√	√
Splint	开放源码	基于程序注解	√		√	√
ARCHER	开放源码	基于调用图	√	√	√	
UNO	开放源码	基于控制流图	√		√	

2.5　动态测试

静态分析无法检测出所有缓冲区溢出问题，因而需要动态测试作为互补的检测方法，增强对程序的安全检测。

目前，检测缓冲区溢出的动态测试主要采用白盒测试方法，即在源程序中插装检测代码实施检测，如Purify[20]和FIST[21]。还有一些测试工具采用输入检测方法进行测试，如Fuzz[22]通过输入伪随机产生的长字符串检测Unix程序的可靠性。

文献[23]将整数范围检测方法应用于动态检测中，开发出动态测试工具STOBO。另外，基于属性的测试方法结合了静态分析和动态测试对缓冲区溢出进行检测。动态测试的检测效果取决于输入数据能否激发缓冲区溢出。理想的输入组合能激发所有潜在的问题，但是其设计非常困难，因为没有一个衡量尺度能说明实验中的输入与理想状态还有多少差距。

3　缓冲区溢出攻击的抵御

目前，预防措施还不能完全避免缓冲区溢出，因而抵御方法是防御缓冲区溢出攻击的另一个重要方面。一旦隐藏在系统中的缓冲区溢出隐患被利用于攻击，这些措施将攻击造成的危害控制在一定范围内。

3.1　抵御方法

1) 基于操作系统的保护。如Linux的内核补丁程序kNoX[24]、PaX[25]实现了不可执行堆栈/堆，阻止了需要在堆栈/堆上执行程序的攻击。PaX通过将函数库随机映射到内存空间，增加了对Return-to-Libc[26]攻击的抵御能力[27]。

2) 基于硬件的保护。Intel/AMD 64位处理器引入NX/AVP的新特性，与操作系统配合可以有效地提高系统的安全性。这种方法存在如无法抵制“借用代码块”攻击[28]等弱点。

3) 返回地址的完整性保护。通过把堆栈上的返回地址保存/备份到一个不受缓冲区溢出影响的空间，确保返回地址不被破坏。目前，这种方法有以下三种实现方式：(1) 扩展编译器[29]。(2) 修改call指令。(3) 利用返回地址栈(RAS)[30]。这种方法的弱点是仅能抵御利用返回地址改变控制流程的攻击。

4) 基于检测值(Canary-based)的监测。通过监测置于缓冲区和被保护数据之间的检测值，检测缓冲区溢出是否发生，例如StackGuard[31]、ProPolice、Libverify[32]以及Visual C++.NET的GS选项。这种方法无法抵御绕过检测值进行的缓冲区溢出攻击。

5) 函数库Libsafe[32]修改了不安全C的标准库函数，加入缓冲区边界检查，检测在调用这些库函数时是否发生缓冲区溢出。

6) 入侵检测工具可以检测包括缓冲区溢出攻击在内的多种攻击，有助于阻止攻击的进一步发展。

7) 访问控制机制，如砂箱(Sandbox)技术限制了不可信应用程序能访问的资源，有利于抵制攻击。

8) 权限最小化原则。鉴于许多攻击都是借助被攻击程序具有的系统级权限获得控制权，遵循权限最小化原则运行程序，可以有效地降低攻击的破坏程度。

3.2 攻击后的恢复

为了阻止攻击的进一步发展，目前缓冲区溢出防卫工具的普遍做法是中止当前的运行程序。这种处理方式简单、直接，但是牺牲了程序的可用性(Availability)。对于服务关键性(Service-critical)程序，中止服务会将缓冲区溢出攻击转换成(Denial of Service，DoS)攻击。文献[33]提出一种编译器扩展方法，将溢出缓冲区的数据保存到一个哈希表中，使其不造成数据破坏，又可供需要时使用，巧妙地化解了可用性与安全性之间的矛盾。文献[34]提出一种处理方法，将程序中的每个函数看成一个交易(Transaction)，当检测到溢出操作时，仅中止当前交易(函数)，而不中止整个程序。

由于程序可用性不容忽视，以后研制的缓冲区溢出防卫工具除了阻止攻击以外，还需要具备一定的恢复能力，在确保安全的同时尽可能减少对程序可用性的影响。

4 讨 论

近十年来涌现出许多针对缓冲区溢出问题的防御策略，然而缓冲区溢出问题然是目前棘手的安全隐患，主要原因包括：(1) 现有策略还不能彻底、有效地解决缓冲区溢出问题。(2) 攻击技术还在不断发展，一些新的攻击利用现有防御方法的弱点实现缓冲区溢出攻击。

今后在探索更有效的解决方法时有以下五个方面值得研究：(1) 普及安全编程教育，降低新开发程序引入安全隐患的可能性。(2) 静态分析方法之间的比较与综合，研究如何进一步减少静态模拟环境与实际运行环境之间的差异，提高静态分析效率和自动化分析程度。(3) 对于静态分析与动态检测有效结合的研究。(4) 减少抵御方法的前提假设，扩大攻击的检测面；降低防卫工具对程序可用性影响的研究。(5) 操作系统与硬件具有的用于抵御攻击的新特性相结合，实现多层面的防御系统。

参 考 文 献

[1] Aleph One. Smashing the stack for fun and profit[EB/OL]. http://www.insecure.org/stf/smashstack.html, 1996-11-19.

[2] LHEE K, CHAPIN S J. Buffer overflow and format string overflow vulnerabilities[J]. Software Practice and Experience, 2003, 33: 423-460.

[3] SPAFFORD E H. The internet worm program: analysis[R]. Purdue University: Technical report CSD-TR-823, 1988.

[4] PINCUS J, BAKER B. Beyond stack smashing: recent advances in exploiting buffer overruns[J]. IEEE Security and Privacy, 2004, 2(4): 20-27.

[5] DEAN D, FELTEN E E, WALLACH D S. Java security: from HotJava to netscape and beyond[C]//In Proceedings of the IEEE Symposium on Security and Privacy. Washington D.C., USA: IEEE Computer Society, 1996:125-130.

[6] JONES R W M, KELLY P H J. Backwards-compatible bounds checking for arrays and pointers in c programs[C]//In Proceedings of the third International Workshop on Automatic Debugging. Sweden: Linkoping University Electronic Press, 1997: 12-26.

[7] JIM T, MORRISETT G, GROSSMAN D, et al. Cyclone: a safe dialect of c[C]//In Proceedings of USENIX Annual Technical Conference. Monterey: USENIX Press, 2002: 275-288.

[8] VIEGA J, BLOCH J T, KOHNO T, et al. ITS4: a static vulnerability scanner for C and C++ code[C]//In Proceedings of the 16th Annual Computer Security Applications Conference. Washington D.C., USA: IEEE Computer Society, 2000: 159-164.

[9] WAGNER D, FOSTER J S, BREWER E A, et al. A first step towards automated detection of buffer overrun vulnerabili-ties[C]//In Proceedings of the seventh Network and Distributed System Security Symposium. San Diego:

Internet Society Press, 2000: 122-127.

[10] GANAPATHY V, JHA S, CHANDLER D, et al. Buffer overrun detection using linear programming and static analysis[C]//In Proceedings of the ACM Conference on Computer and Communication Security. New York, USA: ACM Press 2003: 345-354.

[11] EVANS D, GUTTAG J, HORNING J, et al. LCLint: a tool for using specifications to check code[C]//In Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA: ACM Press, 1994: 87-96.

[12] LAROCHELLE D, EVANS D. Statically detecting likely buffer overflow vulnerabilities[C]//In Proceedings of the 10th USENIX Security Symposium. Monterey: USENIX Press, 2001: 177-190.

[13] DOR N, RODEH M, SAGIV M. CSSV: Towards a realistic tool for statically detecting all buffer overflows in C[C]//In Proceedings of the ACM Press Conference on Programming Language Design and Implementation (PLDI). New York, USA: ACM Press, 2003:155-167.

[14] XIE Y, CHOU A, ENGLER D. Archer: using symbolic, path-sensitive analysis to detect memory access errors[C]//In Proceedings of the 10th ACM press SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA: ACM Press, 2003: 327- 336.

[15] HOLZMANN G. Static source code checking for user-defined properties[C/OL]//In 8th World Conference on Integrated Design and Process Technology. http://citeseer.ist.psu.edu/holzmann02static.html, 2002.

[16] KRATKIEWCIZ K, LIPPMANN R. Using a diagnostic corpus of C programs to evaluate buffer overflow detection by static analysis tool[C/OL]//In 2005 Workshop on the Evaluation of Software Defect Detection Tools. http://www.ll.mit.edu/IST/pubs/050610_Kratkiewicz.pdf, 2005.

[17] XU Z, MILLER B, REPS T. Safety checking of machine code[C]//In SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2000: 70-82.

[18] ZITSER M, LIPPMANN R, LEEK T. Testing static analysis tools using exploitable buffer overflows from open source code[C]//In Proceedings of the 12th ACM SIGSOFT 12th International Symposium on Foundations of Software Engineering. New York, USA: ACM Press, 2004: 97-106.

[19] WILANDER J, KAMKAR M. A comparison of publicly available tools for static intrusion prevention[C/OL]//In Proceedings of the 7th Nordic Workshop on Secure IT Systems. http://citeseer.ist.psu.edu/wilander02comparison.html, 2002.

[20] HASTINGS R, JOYCE B. Purify: Fast detection of memory leaks and access errors[C/OL]//In Proceedings of the Winter USENIX Conference. http://citeseer.ist.psu.edu/291378.html, 1992-05-07.

[21] GHOSH A K, O'CONNOR T, MCGRAW G. An automated approach for identifying potential vulnerabilities in software[C]//In Proceedings of the 1998 IEEE Symposium on Security and Privacy. Washington D.C., USA: IEEE Computer Society, 1998: 104-114.

[22] MILLER B P, KOSKI D, LEE C P, et al. Fuzz revisited: a re-examination of the reliability of UNIX utilities and services[R]. University of Wisconsin: Technical Report CS-TR-95-1268, 1995.

[23] HAUGH E, BISHOP M. Testing C programs for buffer overflow vulnerabilities[C]//In Proceedings of the Symposium on Networked and Distributed System Security. San Diego: Internet Society Press, 2003: 24-30.

[24] ISEC Security Research. Knox-implementation of non-executable page protection mechanism[EB/OL]. http://www. isec. pl/projects/knox/knox.html, 2001-05-17.

[25] The Pax Team. The PAX project[EB/OL]. http://pax.grsecu-rity.net/docs/pax.txt, 2003-11-23.

[26] WOJTCZUK R. Defeating solar designer non-executable stack patch[EB/OL]. http://www.securityfocus.com/archive/1/8470, 1998-05-21.

[27] KIRIANSKY V, BRUENING D, AMARASINGHE S. Secure execution via program shepherding[C]//In Proceedings of the 11th USENIX Security Symposium. Monterey: USENIX Press, 2002: 191-206.

[28] KRAHMER S. X86-64 buffer overflow exploits and the borrowed code chunks exploitation technique[EB/OL]. http://www.suse.de/~krahmer/no-nx.pdf, 2005-03-18.

[29] CHIUEH T C, HSU F H. RAD: a compile-time solution to buffer overflow attacks[C]//In Proceedings of the 21th International Conference on Distributed Computing Systems. Washington D.C., USA: IEEE Computer Society, 2001: 45-50.

[30] YE D, KAELI D. A reliable return address stack: micro-architectural features to defeat stack smashing[J]. ACM SIGARCH Computer Architecture News, 2005, 33(1): 73-80.

[31] COWAN C, PU C, MAIER D, et al. Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks[C]//In Proceedings of the 7th USENIX Security Symposium. Monterey: USENIX Press, 1988: 63-78.

[32] BARATLOO A, SINGH N, TSAI T. Transparent run-time defense against stack smashing attacks[C]//In Proceedings of 9th USENIX Security Symposium. Monterey: USENIX Press, 2000: 18-23.

[33] RINARD M, CADAR C, ROY D, et al. A dynamic technique for eliminating buffer overflow vulnerabilities (and other memory errors)[C]//In Proceedings of the 20th Annual Computer Security Applications Conference. Washington D.C., USA: IEEE Computer Society, 2004: 82-90.

[34] SIDIROGLOU S, GIOVANIDIS G, KEROMYTIS A. Using execution transactions to recover from buffer overflow attacks[R]. Columbia University: Technical Report CUCS-031-04, 2004.

编 辑 黄 莘