

针对非控制数据的缓冲区溢出保护程序

彭贺峰,何 丰

(北方民族大学 计算机科学与工程学院,宁夏 银川 750021)

摘 要:已有的检测缓冲区溢出漏洞的方法有静态的也有动态的。静态分析在软件运行前,析其源代码,找出可能存在的漏洞;动态方法在运行时对可能存在漏洞的软件行为进行监视,发现异常后,进行判断,然后做出适当处理。在分析了传统缓冲区溢出方法的基础上,依据缓冲区溢出攻击的发展趋势,针对非控制数据的缓冲区溢出攻击,提出了一种主要针对非控制数据缓冲区溢出攻击的测试方法,使用变量标识来测试是否发生了缓冲区溢出攻击。这种方法综合了静态和动态分析的优点,能够有效地防御缓冲区溢出攻击。

关键词:缓冲区溢出;非控制数据;控制数据

中图分类号:TP309

文献标识码:A

文章编号:1673-629X(2011)12-0167-05

Protection Program of Buffer Overflow for Non-Control Data

PENG He-feng, HE Feng

(School of Computer Science and Engineering, The North University for Ethnicity, Yinchuan 750021, China)

Abstract: The existing method of detecting buffer overflow vulnerabilities, is static and dynamic. Static analysis before running the software is to analyze the source code, to identify possible vulnerabilities; Dynamic method monitors software behavior at run time and found that exception may be vulnerable, to judge the results and then make the appropriate treatment. It analyzes the traditional method of buffer overflow, according to the trend of buffer overflow attacks, presents a test method that aims at against the non-control data buffer overflow attack, use variables identified to test whether there has been buffer overflow attacks. This method combines the advantages of static analysis and dynamic analysis, can be an effective mechanism against buffer overflow attacks.

Key words: buffer overflow; non-control data; control data

0 引言

现存的检测缓冲区溢出漏洞的方法,有静态的也有动态的。静态分析在软件运行前^[1],分析其源代码,找出可能存在的漏洞;动态方法在运行时对可能存在漏洞的软件行为进行监视^[2],发现异常后,进行判断,然后做出适当处理。

以应用程序控制数据为目标的代码注入^[3]攻击已经繁荣了20多年^[4],现代操作系统已经开发了多种多样的抵御方法,各种各样的漏洞补丁层出不穷,对攻击者来说,攻击的难度也在逐渐增加,对于目前这种趋势,漏洞利用者正逐渐把目标从控制数据转向非控制数据,对于非控制数据攻击,攻击者改变了程序逻辑上使用的重要数据结构,这样就使本来以预定路线执行的程序发生了改变,比如,溢出一个表征当前用户是否是管理员的布尔变量并且设置该变量为“true”,由此

可以获得访问程序的管理员功能。文中提出一种基于标识值的防御机制,通过该机制来保护应用程序不受非控制数据缓冲区溢出^[5-7]攻击,通过在所有变量之前插入标识值,并在每次使用变量时验证标识与变量的完整性,如果发生改变变量内容的缓冲区溢出,就会导致变量的特征值也发生了改变,这样就能够检测到发生了溢出攻击,称之为标识保护。标识保护可以作为一个测试工具用于将要开发完成的应用程序,也可以对程序的高危部分提供实时保护。除了程序被保护部分之外,对程序的其他部分不会产生额外的时间损耗。

1 相关技术

缓冲区溢出已经成为最广为人知的程序漏洞之一,20年来一直被攻击者广泛利用,前面提到过的莫里斯蠕虫、红色代码^[8]、SQL Slammer都主要利用了缓冲区溢出方法攻击并感染新的宿主,如今,虽然对缓冲区溢出的原因已有了本质的了解,同时也实施了一些解决方案,但很多年过去了,缓冲区溢出依然像瘟疫一样肆虐在各种由不安全的语言编写的软件领域,SANS

收稿日期:2011-04-26;修回日期:2011-07-28

基金项目:国家自然科学基金(71061001/G011201)

作者简介:彭贺峰(1977-),男,河南人,硕士研究生,研究方向为数据挖掘;何 丰,教授,主要从事语义 web 和数据挖掘的研究。

应用安全论坛把“classic buffer overflow”在 25 个最危险的程序缺陷排名中放在第 3 位。

缓冲区溢出漏洞攻击者常常选择把攻击代码放置在脆弱程序的变量中,使用溢出变量本身覆盖的内存位置,来控制正在运行的程序的控制流。这些溢出的目标地址通常是返回地址、保存的基址指针、函数指针等等。攻击者的目标是控制应用程序执行权。为此,研究者和编程人员也在尽力保护应用程序的控制数据(control data),前面提到过的 Stackguard^[9]和 DEP^[10],就是 2 种在现代操作系统中广泛使用的用来抵御控制数据攻击的方法,并专门设计用来抵御控制数据攻击。前者在每个栈帧的函数返回地址前放置标识并在函数被允许返回前检查标识的完整性来抵御溢出攻击。后者通过使正在运行进程的堆栈内存页面不可执行来阻止攻击者,即使攻击者获得进程控制权,也不能够执行已经注入的代码。

防御技术的发展,使得成功的实现控制数据溢出越来越难,攻击者逐渐将关注的目光转移到新的溢出技术,通过新技术依然能够取得相同的效果,这种新技术就是非控制数据攻击,非控制数据攻击使得攻击者不再需要注入和执行自己的代码,只要识别出目标程序中存在的可利用部分(如允许以管理员身份运行的函数),然后通过改变程序数据结构中的值,就可以访问未授权访问的函数,已有的一些对控制数据攻击有效的方法不能检测非控制数据攻击(包括上述提到的 Stackguard 和 DEP)。

文中提出一种抵御非控制数据攻击的方法,通过对程序源代码中的所有变量做标识,然后通过赋给随机值使这些标识保持彼此独立。一个变量前面放置一个唯一标识,攻击者如果使用缓冲区溢出改变一个变量的内容,在覆盖变量自身之前不可避免地会覆盖它的标识,只要在使用变量前检测变量标识值的完整性,如果发现异常,强制终止进程,这样就有效地防止了非控制数据攻击。

可以根据应用程序的重要性来使用标识保护,也可以把它作为一个测试工具来发现隐含漏洞,或者作为有效的阻止非控制数据缓冲区溢出攻击的实时检测工具,在测试该方法的有效性时,发现了以前用其它检测工具检测时没有报告的一个基于堆的缓冲区溢出漏洞。

2 非控制数据

这里介绍被非控制数据攻击作为目标的不同的数据结构,然后给出一个攻击实例。

2.1 临界数据结构

一些研究者指出非控制数据攻击和控制数据攻击

一样危险,他们对应用程序测试后显示:试图进行非控制数据攻击的攻击者,为了覆盖正在运行的应用程序,必须处理一些临界数据结构,研究者将这些数据结构归类为四种不同类型:

(1) 配置数据。

存储在进程内存中的数据,比如从一个配置文件读入到进程内存中的数据,进程认为该数据属于系统管理员组,如果攻击者能够覆盖该数据,进程将会以管理员不可预见的方式运行。

(2) 用户标识数据。

用户身份识别数据,用户登录后,常常用该数据进行资源访问,如果数据被替换,攻击者能够模仿其它用户的身份对该用户资源进行非授权^[11]访问。

(3) 用户输入字符串。

用户输入符合处理程序格式要求的信息并确认后,程序正在处理的过程中,如果攻击者能够冒充合法用户输入字符串,程序会认为这是一个安全的输入,虽然它并不是安全的。

(4) 决策数据^[12]。

覆盖用于做决策的数据明显会带来灾难性的后果,在 2.2 节的例子就以决策数据为目标。

显然地,任何应用程序都会包含上述数据结构的一个子集,与控制数据不同,非控制数据攻击者至少需要了解程序部分语义,一些研究者已经对此做了证明,非控制数据攻击已经称为一种软件领域面临的威胁,因为独特的研发策略而不能被控制数据攻击者利用的程序,对于非控制数据攻击来说可能是脆弱的。

2.2 非控制数据攻击

非控制数据漏洞代码示例如下:

```
int main (int argc, char * * argv) {
    char p[40];
    int u=0;
    char b[30];
    char * p;
    readPassFile(PASSFILE, p, sizeof(p));
    printf("Input password: ");
    fgets(b, sizeof(p), stdin);
    if(! strcmp(b, p)) {u=1;}
    if (u) {
        printf("Y\n");
        execl("/bin/sh", "sh", NULL);
    }
    return 0;
}
```

考虑上面的代码,程序的目的是认证用户,如果用户提供正确的口令,能够得到一个 Shell,否则程序结束。main() 函数调用 fgets() 函数读取用户输入文本,

fgets() 认为是安全的函数, 因为第二个参数可以表明要读的最大字符个数, 程序员误用该参数, 用 size of() 作为 fgets() 的第二个参数, 这里它的大小由 size of(p) 提供。因此 fgets() 最多可以读 40 个字符, 比缓冲区多 10 个。

当本程序和栈溢出保护一起编译后, 漏洞不能用来做控制数据攻击, 可是, 当缓冲区变量溢出后, 因为两个变量在栈中相连存储, u 被覆盖, 该变量本来由程序在认证成功时设置, 攻击者溢出该变量并把它设置为非 0 值, 程序会认为认证成功(无论其是否成功), 然后去执行/bin/sh shell。

3 设计

基于前面介绍的 StackGuard 的基本理念, 然后由只对返回地址保护覆盖到所有变量, 显然, 这样会导致很高的性能额外开销, 但无疑是一种有效的检测发生在堆栈段某些部分的错误和攻击的方法。

程序汇编期间, 使用标识保护框架重写应用程序源代码并且把所有变量封装到保护结构中, 保护结构由 C struct 实现, 包括 2 个项, 原始变量和标识值, 当一个变量被分配, 无论是在栈还是堆, 其标识值都初始化为一个随机值, 对每一个应用程序的运行变化, 进一步检测每个变量的使用, 检查标识完整性并额外地插入相应的标识。

检测通过指针间接使用的变量需要指针支持, 下图显示对一个应用程序的“重要”变量的验证变得复杂, 原因是它通过指针进行访问, 图 1 显示了程序正常运行时栈的情况, 如果攻击者设法通过调用 strcpy 函数去覆盖“important”变量的值(包括标记), 变量的变化不会被检测到, 因为当指针变量‘p’解除引用后, 进行检测, 验证‘p’的标识值在攻击时并没有发生变化, 实际上‘p’引用的变量发生了变化, 见图 2。

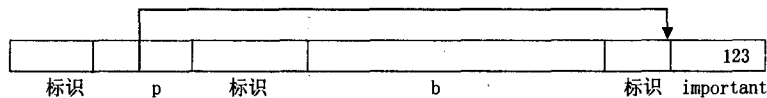


图 1 程序正常运行时的栈

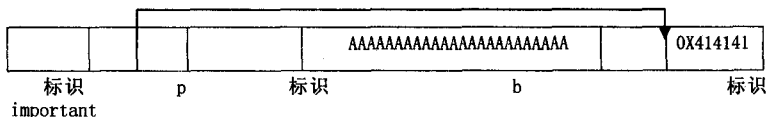


图 2 恶意的 strcpy() 函数使用后的栈

问题的根源是没有首先对该变量前的标识值进行完整性检测, 直接通过指针引用该变量, 为此, 要检测每个解除引用, 查找并验证变量被解除引用前的标识, 由于编译时指针指向的目标是不可预知(不可决定)的, 所以查找是必要的。

标识保护使用一个内存映射^[13]来存储内存空间

中所有注册对象信息, 当对象创建时, 它们在内存映射中注册, 对于每一个指针解除引用动作, 找到对应的内存对象并且对其对应的标识进行验证。

一段显示指针解除引用使得标识验证复杂化的代码如下:

```
int main (int argc ,char * * argv){
    int important = 123;
    int b[80];
    int * p=&important;
    strcpy(b,argv[1]);
    printf("%d\n", *p);
    return 0;
}
```

并不是对所有的进程中的代码都能够这样做, 比如一些共享库中的代码就对标识的使用透明(看不到), 和其他类似方法不同, 标识保护不改变指针或者函数的传统调用的表示方法, 和已有程序代码保持了完全的兼容性。这同时也表明了标识保护支持可选择的使用, 程序开发者可以只选择保护应用程序中关键的部分来减少额外开销。

标识保护不改变编程者定义的结构布局, 一些编程者依赖内存中对象确切的布局, 为此, 不能够在一个结构的不同区域之间插入标识来破坏内存布局的完整性, 这样导致了不能对单一存储结构内缓冲区溢出的数据进行检测, 已有的大多数防御机制都有这种局限性。

4 原型实现

4.1 标识

通过使用 CIL^[14]架构转换 C 代码来实现标识保护, 在实现的过程中, 通过一个自定义的 CIL 插件对代码进行修改。

标识由整型变量表示, 和所保护的变量封装在一起, 标识的随机值数组在程序开始时初始化, 每一个标识初始化为该表中的值, 标识值进入表中的顺序由编译时确定, 代码转换的时候, 对标识的验证调用插入在标识所保护的变量声明之前, 如果一个验证失败, 以段错误来强制终止

程序的运行。

笔者做了一些优化措施来降低额外开销, 首先, 安全的变量分在一组, 由一个单一的标识来保护, 所谓安全的变量就是那些地址重来不用的变量, 数组和用取地址符号(&)使用的变量是不安全的; 其次, 多个标识验证调用可以顺序进行, 多个对同一标识的验证, 显然可以集中在一起, 共同使用一个验证调用; 第三, 之前

由安全语句处理的验证调用向上移,以便集中在一起,所谓安全就是不会威胁到任何标识的完整性(如赋给一个变量计算结果);最后,对于安全的函数不再嵌入额外代码。安全的函数指只使用安全的语句和局部变量的函数。

4.2 内存映射

内存映射中存储了内存对象的起始地址,以 2^k bytes 为一块分配在内存空间中,内存映射用一项用来保存对象的起始地址,内存对象必须对齐存放且是 2^k bytes 的整数倍,为了处理堆中的内存对象映射,内存分配函数 malloc, calloc, realloc 和 free 由一个包装函数进行重写,包装函数对标识值分配额外的内存、初始化标识,并登记内存对象到内存映射中,在每一个指针解除引用之前,插入一个附加的验证调用,在内存中查找解除引用的指针,如果它指向一个受保护的對象,验证该对象标识的完整性,附加的验证与正常的标识验证相同,如果验证标识失败,同样终止进程的运行。

5 安全评估

本节评估标识保护能够提供的安全性,举出例子展示标识保护如何检测现实中的攻击。

5.1 有效性

标识保护检测数据缓冲区溢出的有效性只在于对修改后的标识值精确的检测。正如第 4 节所解释:一个变量的标识是被保护程序在运行时随机选择的整型变量数字,利用缓冲区溢出而进行非控制数据攻击的攻击者,为了躲避检测,必须能够恢复到原来的标识值,主要通过 2 种方法做恢复初始的标识值,一是暴力破解^[15],二是通过内存泄露攻击找出标识的值。

暴力破解:标识保护检测到一个不完整的标识时,就会马上终止正在运行的进程,对于一个 4 字节的标识(标准整型变量),最坏情况下,攻击者需要做 2^{32} 次尝试才能够找到正确值,据此,在攻击者成功之前,进程将会终止 $2^{32}-1$ 次,相信系统管理员能够在 4 亿次尝试结束前及时地发现发生的攻击。

内存泄露: Raoul Strackx 已经展示了特定的编程错误如何会暴露部分内存给攻击者,攻击者可以对泄露内存使用依靠隐蔽数据的逆随机方法,获得内存中的信息。虽然这种攻击是可能的,但是我们认为,对标识检测方法来说,这是一种不太可能发生的情况,攻击者必须找到所溢出变量的标识,而不仅仅是任意一个隐蔽的标识。因为标识保护使用多个随机标识,从而一个标识妥协并不一定会导致整个方法妥协。

实践证明通过 32 位数据提供的随机数能够有效地保证安全性,一些人会提出异议:边界检测器提供更好的安全性,因为它们的检测和随机数无关。必须指

出的是,标识保护的方法可以检测发生在第三方代码(比如库)的缓冲区溢出,而边界检测却不能,边界检测器只能通过对它已经加工过的代码来检测溢出,因此,当第三方代码(如外部库)访问变量时,边界检测器起不到保护作用。我们的方法能够检测溢出的发生,这是由于标识的改变和溢出发生的位置无关。

5.2 测试

使用包含非控制数据攻击漏洞的程序对标识保护进行测试,经过多种攻击方法尝试后,程序没有被攻破。同时,发现攻击不成功的主要原因是因为攻击者不能破解由随机数生成器生成的 32 位标识值。

5.3 em3d 中的堆溢出

由 Olden 基准测试^[16]组件对标识保护在 em3d 中进行测试时发现了一个堆溢出漏洞,该漏洞由 make_graph.c 中的 initialize_graph 函数产生:赋值 `retval->e_nodes[i]=local_node`;执行循环 `for i=1 to NumNodes`, NumNodes 是一个命令行参数,但 `retval` 中的 `e_nodes` 数组只分配了固定大小值(由包含在 em3d.h 中的 PROCS 常数决定)的空间。

这个堆溢出的发现进一步验证了标识保护能够检测并终止非控制数据攻击,据了解,之前没有其他的防御机制检测到该溢出。

6 结束语

还有一些别的保护缓冲区溢出攻击的方法,只简单讨论比较重要的几种。

一些方法在抵御缓冲区溢出漏洞时使用随机数, Canary-base 方法使用一个存储在重要内存位置前的秘密随机数;如果该随机数在执行一些操作之后发生了改变,就可以检测出发生了攻击。内存迷惑方法使用随机数加密^[17,18](常常通过异或操作)重要内存地址或者别的重要信息。内存布局随机发生器随机内存布局:在随机地址载入堆和栈并且在二者之间放置随机间隔。指令集随机发生器加密内存中的指令并在执行之前对其进行解密。

这些方法通过加密内存位置,效果虽然很好,然而包含内存溢出的程序依然包含“内存重读”(比如使用 `strcpy` 复制没有用 NULL 结尾的字符串能够使内存信息泄露)或者别的像格式化字符串^[19,20]漏洞,可以被攻击者用来输出内存位置,内存泄露漏洞依然能够使黑客攻陷这种类型的保护方法。

参考文献:

- [1] 苏璞睿,杨 轶. 基于可执行文件静态分析的入侵检测模型[J]. 计算机学报, 2006, 29 (9): 1572-1578.
- [2] 官传平,官云战. 数组越界的静态测试分析[J]. 计算机工

- 程,2006(3):70-72.
- [3] 朱若磊. 利用核心态钩挂技术防止代码注入攻击[J]. 计算机应用,2006,26(9):2134-2136.
- [4] 余俊松,张玉清,宋 杨,等. Windows 下缓冲区溢出漏洞的利用[J]. 计算机工程,2007,33(17):162-164.
- [5] 苏 朋,陈性元,唐慧林. Windows 缓冲区溢出 Exploit 代码分析研究[J]. 计算机安全,2008(1):48-52.
- [6] 王志飞. Windows 平台下缓冲区溢出的攻击及防卫[J]. 辽宁师专学报(自然科学版),2005(2):12-17.
- [7] Avijit K, Gupta P, Gupta D. TIED, LibsafePlus: Tools for runtime buffer overflow protection [C]//Proceedings of the 13th Conference on USENIX Security Symposium. Berkeley: Usenix Association,2002:191-206.
- [8] 张小斌,严望佳. 黑客分析与防范技术[M]. 北京:清华大学出版社,1999:97-104.
- [9] Rozinov K. Stackguard Protecting Against Buffer Overflows Part I - Detailed Overview [D]. [s. l.]:Polytechnic University,2003.
- [10] 大 熊. Windows 内置的病毒防护-DEP[J]. 电脑爱好者,2005(7):86-87.
- [11] 赵 磊,蒋本伦,朱玉龙. 无线局域网非授权访问控制技术[J]. 今日科苑,2008(15):108-108.
- [12] Dong Guozhu, Su Jianwen. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL * [J]. ACM SIGMOD Record,2000,29(1):44-51.
- [13] Waldspurger C A. Memory resource management in VMware ESX server [C]//In:Proceedings of 5th Symposium on Operating Systems Design and Implementation(OSDI). New York: ACM,2002:181-194.
- [14] Poon E, Fleet D J. Hybrid Monte Carlo Filtering: Edge-Based People Tracking [J]. IEEE Motion and Video Computing, 2002,5(11):151-158.
- [15] Oechslin. Making a faster cryptanalytic time-memory trade-off [C]// Proc. CRYPTO'03. [s. l.]:Springer,2003:617-630.
- [16] ORIGIN2000 服务器 SPEC 基准测试性能创世界记录[J]. 计算机辅助设计与制造,1997(7):58-58.
- [17] Craig J C, Webb J. Microsoft Visual Basic6.0 程序开发环境 [M]. 北京:博彦科技发展有限公司,2000:320-326;625-635.
- [18] 吴业福. 用 VB6 实现汉字的加密方法探讨[J]. 计算机应用研究,2001(3):143-145.
- [19] Wilander J, Kamkar M. Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention [C]//10th Network and Distributed System Security Symposium. [s. l.]:[s. n.], 2003:149-162.
- [20] Ringenburt M, Grossman D. Preventing format-string attacks via automatic and efficient dynamic checking [C]//Proceedings of the 12th ACM Conference on Computer and Communications Security. [s. l.]:[s. n.],2005:354-363.

(上接第 166 页)

安全指南》为接触云服务提供商的企业提供有效的信息安全对策,CSA 成员仍在继续云信息安全对策的研究。

从管理角度看,随着云计算的发展,NIST、ISACA 均提倡用管理的手段控制和减小云计算安全风险。云计算服务级别协议(SLA)是云计算服务提供商和云服务用户间唯一的法律协议^[11],云服务提供商通过 SLA 获得用户信任,考虑到云计算和大型主机时代之间的相似性(对远程资源的高度依赖),云计算服务级别协议(SLA)的广泛使用势在必行^[12]。

5 结束语

云计算是近年来的研究热点,文中分析了云计算环境下存在的信息安全问题,随着云计算的进一步发展和应用,信息安全问题势必成为云计算发展的关键技术问题,在这方面的研究还有很长的路要走。

参考文献:

- [1] 赵 粮,裴晓峰. 云计算环境的安全威胁和保护[J]. 中国计算机学会通讯,2010,6(5):47-50.
- [2] CSA. Security Guidance for Critical Areas of Focus in Cloud Computing V2. 1 [EB/OL]. [2010-05-10]. <http://www.cloudsecurityalliance.org/guidance/>.
- [3] Forum J. Cloud cube model: selecting cloud informations for secure collaboration [EB/OL]. [2010-05-10]. http://WWW.opongroup.org/Jericho/eloud_cube-model_0.pdf.
- [4] McCalline S, Jacobson V. The BSD Packet Filter: A New Architecture for User-level Packet Capture [C]//Proceedings of the 1993 Winter USENIX Technical Conference. San Diego, CA:USENIX,1993.
- [5] Miller M. 云计算[M]. 姜进磊,孙瑞志,向 勇,等译. 北京:机械工业出版社,2009.
- [6] 张为民,唐剑峰,罗治国,等. 云计算深刻改变未来[M]. 北京:科学出版社,2009.
- [7] 叶 伟. 互联网时代的软件革命-SaaS 架构设计[M]. 北京:电子工业出版社,2009.
- [8] 李德毅. 云计算:从图灵计算到网络计算[C]. 2009 云计算中国论坛. 出版地不详:出版者不详,2009.
- [9] 王 鹏,黄华峰,曹 珂. 云计算:中国未来的 IT 战略 [M]. 北京:人民邮电出版社,2010.
- [10] IBM 虚拟化与云计算小组. 虚拟化与云计算[M]. 北京:电子工业出版社,2009.
- [11] ISACA. Cloud Computing: Business Benefits With Security, Governance and Assurance Perspectives [EB/OL]. [2010-05-10]. <http://WWW.isaca.org/Template.cfm?Section=Nederlands&Template=/search/SearchDisplay.cfm>.
- [12] 张云勇,陈清金,潘松柏,等. 云计算安全关键技术分析 [J]. 电信科学,2010(9):64-69.