ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.11, November 2010, pp.2944−2958
doi: 10.3724/SP.J.1001.2010.03672

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于异常捕获的强脆弱性分析和利用*

时云峰[1+], 张金祥[2], 冯建华[1]

[1](清华大学 计算机科学与技术系,北京 100084)

[2](信息安全测评认证中心,北京 100016)

Critical Vulnerability Analysis and Exploitation Based on Exception Capture

SHI Yun-Feng[1+], ZHANG Jin-Xiang[2], FENG Jian-Hua[1]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](Information Security Test and Evaluation Center, Beijing 100016, China)

+ Corresponding author: E-mail: noble_shi@126.com

Shi YF, Zhang JX, Feng JH. Critical vulnerability analysis and exploitation based on exception capture. *Journal of Software*, 2010,21(11):2944−2958. http://www.jos.org.cn/1000-9825/3672.htm

Abstract: In this paper, critical vulnerability is parsed from its essence, analysis and exploitation. First, this paper gives the definition of critical vulnerability, present necessary and sufficient condition of the existence for critical vulnerability, and proves that there are not any universal detecting procedures for critical vulnerability. Secondly, this paper proposes three basic conditions to judge if a procedure has critical vulnerability, examines the essential method to analyze critical vulnerability using the backtracking analysis, and proves that the time complexity of the backtracking analysis conforms with the exponential growth of at least $O(2^h)$. Lastly, this paper ascribes the critical vulnerability exploitation to solving critical vulnerability equation sets, and gives the algorithm for solving the critical vulnerability equation set by a generalized equation and VC factorization. Then, the paper analyzes and computes two critical vulnerabilities of the Office series software.

Key words: critical vulnerability; backtracking analysis; relatively control; generalized equation; VC factorization

摘 要: 从强脆弱性的本质、分析和利用 3 个方面对强脆弱性进行了剖析.首先给出了强脆弱性定义,提出了程序存在强脆弱性的必要条件和充分条件,并证明了不存在万能的强脆弱性检测程序;其次,提出了判断程序存在强脆弱性的 3 条基本依据和利用回溯分析进行强脆弱性分析的基本方法,证明了回溯分析时间复杂度符合指数函数增长规律且至少为 $O(2^h)$;最后,将强脆弱性利用归结为强脆弱性方程组求解问题,并给出了利用相对可控、广义方程和 VC 分解对强脆弱性方程组进行求解的相应算法.并且对 Office 系列软件的两个强脆弱性进行了分析和计算.

关键词: 强脆弱性;回溯分析;相对可控;广义方程;VC 分解

中图法分类号: TP309 **文献标识码**: A

强脆弱性通常是指可以用来进行网络(或主机)攻击、并部分或完全控制被攻击系统的软件脆弱性,很多软件强脆弱性利用程序都被用于实施网络(或主机)攻击.如果一个流行软件的未公开强脆弱性利用程序被不怀好

* Supported by the National Natural Science Foundation of China under Grant No.60473083 (国家自然科学基金)
Received 2008-08-28; Revised 2009-04-29; Accepted 2009-07-06

意的人所掌握,意味着一旦满足攻击条件,就可以部分或完全控制任何安装该软件的系统.这对于软件应用者是致命的危害,因此很有必要对软件强脆弱性进行分析和研究.

目前,针对软件脆弱性的研究主要集中在脆弱性检测方面,专门针对强脆弱性理论分析和利用方面的研究还很少.从研究方法上看,程序脆弱性检测有针对源代码的脆弱性检测和针对二进制反汇编的脆弱性检测.前者已经形成工具,如 MemSherlock[1],Ccured[2,3].上述工具可对程序源代码进行自动分析,根据分析结果判定程序是否存在脆弱性.FUZZ[4,5]是一种典型的黑盒测试方法,主要是通过构造可能导致程序出现问题的输入数据进行自动测试,通过 Fuzz Testing 可使程序产生异常[6],这是一种较为成熟实用的软件脆弱性测试方法.缓冲区溢出往往可以导致程序存在强脆弱性,针对缓冲区溢出的研究很多[7,8],大多是通过内存跟踪实现其基本功能,在大型软件中实现内存全面、彻底的长期稳定跟踪还存在一些问题.

程序产生异常(错误)说明其存在脆弱性,但这并不等于该脆弱性可以被成功利用,未必能够编写出用于网络(主机)攻击的强脆弱性利用程序,即不一定存在强脆弱性.本文在软件脆弱性的基础上提出了强脆弱性概念,对强脆弱性进行了形式化定义,证明了程序存在强脆弱性的基本条件,提出了判断程序存在强脆弱性的基本依据和分析强脆弱性的基本方法.当确定程序存在强脆弱性后,将强脆弱性利用条件抽象成强脆弱性方程组,通过对强脆弱性方程组求解达到利用强脆弱性的目的.

本文第 1 节介绍异常的基本概念并剖析强脆弱性的本质,提出程序存在强脆弱性的必要条件和充分条件,并证明不存在万能的强脆弱性检测程序.第 2 节对强脆弱性进行全面分析,提出判断程序存在强脆弱性的 3 条基本依据和利用回溯分析进行强脆弱性分析的基本方法.第 3 节介绍与强脆弱性利用相关的强脆弱性方程组的概念,并给出利用相对可控、广义方程和 VC 分解对其求解的算法.第 4 节析 OFFICE 系列软件存在的两个强脆弱性.第 5 节对文章进行总结.如非特殊说明,本文示例均以 32 位 X86 体系结构汇编指令描述程序流程.

1 强脆弱性本质

在对强脆弱性分析之前,先介绍几个与强脆弱性分析相关的概念:异常、程序连续点和间断点,然后剖析强脆弱性的本质以及程序存在强脆弱性的必要条件和充分条件.

1.1 异常捕获

异常是由于 CPU 执行了某些指令引起的,可以包括存储器存取违规、除 0 或者特定调试指令等,其实系统内核也将系统服务视为异常.下面以 Windows 2000 为例对异常及异常捕获机制进行简要介绍.

当发生异常后,CPU 硬件将控制权传递给内核陷阱处理器,内核陷阱处理器创建一个陷阱帧.正是由于该陷阱帧,在异常被解决后,系统可以从它停止的地方恢复运行.如果异常发生于内核模式,则仅由内核异常处理器来处理该异常,如缺页时通过调用页程序 *MmAccessFault*()将页换入物理内存.如果异常发生于用户模式,则系统首先进行内核态异常处理,内核异常处理建立异常记录 *ExceptionRecord* 并调用函数 *KiDispatchException*()在内核空间进行异常处理.内核异常处理结束后,将控制权交给用户态,用户态第 1 个处理异常的是 *ntdll.dll* 中的 *KiUserExceptionDispatcher*()函数.该函数首先寻找堆栈中基于帧的异常处理例程,即搜索 S.E.H(structed exception handling)链表.若所有的 S.E.H 都失败,或用户根本没有定义进程异常处理,那么系统默认的异常处理函数 *UnhandleExceptionFilter*()将被调用.U.E.F 会根据注册表里的相关信息决定是默默地关闭程序,还是弹出错误对话框[9−12].

所谓异常捕获就是捕获操作系统产生的异常,一般只需截获函数 *KiUserExceptionDispatcher* 即可达到目的.截获 *KiUserExceptionDispatcher* 后可以得到 *ExceptionRecord* 和 *ContextRecord* 结构,这两个结构包含了异常产生的位置、异常代码和产生异常时各个寄存器的值等信息.

1.2 连续点和间断点

定义 1. $[\cdot]_n$ 为取值运算符,$[x]_n$ 表示获取内存空间 x 处存储的 n 字节数据,n 为 4 时可省略下标,常见的 *mov*, *pop* 等指令都可看作取值运算.

将程序执行过程作如下描述:

$$x_{i+1}=\begin{cases}x_i+len(op(x_i)), & op(x_i)\in A\\ f(op(x_i),m_{i1},m_{i2},...,r_{i1},r_{i2},...), & op(x_i)\in B\end{cases},$$

其中,x_i表示当前指令地址,$op(x_i)$表示当前指令,$(m_{i1},m_{i2},...,r_{i1},r_{i2},...)$表示 $op(x_i)$执行所需的内存和寄存器数据,x_{i+1}表示执行完 $op(x_i)$后 CPU 需要执行的下一指令地址,f 表示根据当前参数确定 x_{i+1}数值的计算函数,$A=\{a_1,a_2,...\}$表示顺序指令集合,$B=\{b_1,b_2,...\}$表示跳转指令集合.上述函数表示:

(1) 如果 $op(x_i)$为顺序执行(如 *mov*,*cmp* 等),则执行完 $op(x_i)$后,CPU 需要执行的下一指令为当前指令地址加上当前指令长度.此时称程序在 x_i 处连续,点 x_i 为程序连续点.

(2) 如果 $op(x_i)$为跳转指令(如 *jmp*,*ret*,*call* 等),则执行完 $op(x_i)$后,CPU 需要执行的下一指令由 $op(x_i)$和 $op(x_i)$所需的操作数共同决定.此时称程序在 x_i 处间断,点 x_i 为程序间断点.

定义 2. 程序的间断点可以表示为 *J*+*N*,其中,*J* 表示跳转指令,*N* 表示跳转地址.如果 *N* 为常量,则称为第 1 类间断点(或常量间断点),如 *jmp* 0*xAABBCCDD*;如果 *N* 为变量,则称为第 2 类间断点(或变量间断点),如 *jmp* [*eax*].在 32 位 X86 体系结构中,当程序遇到间断点 *J*+*N* 后的执行过程可作如下描述:

$$x_{i+1}=\begin{cases}N, & J+N\text{为第1类间断点}\\ [N], & J+N\text{为第2类间断点}\end{cases}.$$

1.3 强脆弱性本质

下面给出强脆弱性的确切定义,然后给出程序存在强脆弱性的必要条件和充分条件.

定义 3. 一个程序 *P*,通过输入集 $I=\{i_1,i_2,...\}$,完成的功能集为 $F=\{f_1,f_2,...\}$,则表示为 $F=P(I)$.如果$\exists I'$作用于程序 *P* 满足 $F'=P(I')$且$\exists f'\in F'$,f'功能为控制程序 *P* 的流程到当前进程空间任意位置,则程序 *P* 存在强脆弱性.

上述定义表明,如果程序 *P* 存在强脆弱性,程序流程将受输入数据的影响,而且影响的程度非常强烈,以至于可以控制 EIP 为当前进程空间的任意位置.换句话说,强脆弱性就是指输入数据可以任意控制程序流程.

定义 4. 一个程序 *P* 称为 $\bar{N}$ 位常规程序($\bar{N}=2^m$,*m* 为大于 4 的自然数.如非特殊说明,$\bar{N}$ 均满足此条件),如果:

(1) 程序 *P* 字长为 $\bar{N}$ bit,最大寻址空间和进程线性空间均为 $2^{\bar{N}}$ bit.

(2) 程序 *P* 存在代码空间区域 *C* 和数据空间区域 *D*,其中,

$$C=[c_b,c_e],\ 0\leqslant c_b<c_e\leqslant 2^{\bar{N}},$$

$$D=D_1\cup D_2\cup...\cup D_n=[d_{b1},d_{e1}]\cup[d_{b2},d_{e2}]\cup...\cup[d_{bn},d_{en}],\text{其中},0\leqslant d_{b1}<d_{e1}<d_{b2}<d_{e2}<...<d_{bn}<d_{en}\leqslant 2^{\bar{N}}.$$

(3) *P* 的可执行代码位于代码空间区域 *C*,程序执行所需数据(包括输入数据)位于数据空间区域 *D*.

(4) 存在正常数$\varepsilon_1,\varepsilon_2$满足$[c_b-\varepsilon_1,c_b]$和$[c_e,c_e+\varepsilon_2]$区域均不可执行.

(5) *P* 在不可执行区域内强制执行代码会因触发异常而被动终止.

(6) *P* 存在输入表 *I*–*T*$\in D_i=[d_{bi},d_{ei}]$,*I*–*T* 中存储有其他模块导出的函数地址.

(7) *P* 通过访问输入表 *I*–*T* 获得其他模块导出的函数地址,*P* 通过转移到该函数地址执行代码完成对其他模块导出函数的调用.

(8) *P* 产生异常时必然调用函数 *KiUserExceptionDispatcher*,且 *KiUserExceptionDispatcher* 不属于模块 *P*.

定义 5. 区域 $D_i=[d_{bi},d_{ei}]$称为可控区域,如果对于任意储于 D_i 中的单字节数据 *y* 满足:

(1) $\forall y\in[0,FF],\exists x\in I$ 和函数 *f* 满足 $y=f(x)$.

(2) 若上述 *x* 和 *f* 满足 $y_1=f(x)$且 $y_2=f(x)$,则 $y_1=y_2$.

I 为用户输入数据集合.存储于可控区域 D_i 中的数据称为可控数据.

命题 1. 程序存在强脆弱性的必要条件:如果 $\bar{N}$ 位常规程序 *P* 存在强脆弱性,则 *P* 必然存在间断点.

证明:用反正法.假设程序 *P* 不存在间断点,则 *P* 的所有执行点都是连续点.由于程序 *P* 为 $\bar{N}$ 位常规程序,因此 *P* 可执行代码位于代码空间区域$[c_b,c_e]$,所以 *P* 的程序入口地址必然位于代码空间区域$[c_b,c_e]$.假设程序入口地址为 x_0,则后续指令地址可作如下表示:

$$x_1=x_0+len(op(x_0)),$$

$$x_2=x_1+len(op(x_1)),$$

$$\ldots,$$

$$x_i=x_{i-1}+len(op(x_{i-1})).$$

因此,程序执行会有两种结果:

(1) 程序执行到 c_e 之前主动退出.

(2) 程序在代码段一直连续执行,直到运行到 c_e 仍不主动退出.由于$\exists\varepsilon$满足区域$[c_e,c_e+\varepsilon]$不可执行,因此程序在 c_e 继续执行会因触发异常而被动终止.

因此,无论输入数据集合 I 满足什么条件,程序 P 都不可能被任意改变程序流程.对于 $F=P(I)$来说,不存在 $f\in F$,f 功能为控制程序 P 的流程到当前进程空间任意位置,即程序 P 不存在强脆弱性,与题设矛盾.所以,程序 P 必然存在间断点.证毕. □

该命题指出,如果程序存在强脆弱性,则其内部必然存在跳转指令,如类似 *jmp*,*call*,*ret* 等非连续指令.

在 $\bar{N}$ 位常规程序 P 中,假设 *reg* 表示可容纳 $\bar{N}/8$ 字节的通用寄存器,*data* 表示字长为 $\bar{N}/8$ 字节的立即数,方便起见,对寄存器赋值指令表示为 *mov reg*,*data*,该语句长度表示为 $\bar{N}/8+\bar{M}$ 字节;跳转指令表示为 *jmp reg*,该语句长度表示为 $\bar{J}$ 字节.

命题 2. 程序存在强脆弱性的充分条件 1:如果 $\bar{N}$ 位常规程序 P 存在可使其转移到可控制区域 D_i 的间断点,且区域 D_i 长度不小于 $\bar{N}/8+\bar{M}+\bar{J}$ 字节,则程序 P 必然产生强脆弱性.

证明:由于程序 P 存在可控区域 D_i,因此可构造输入集 I,使其满足在可控制区域 D_i 起始位置存在如下代码(见表 1):

Table 1 Instruction and its corresponding length

表 1 指令语句及其对应长度

Instruction	Instruction length
mov reg, data	$\bar{N}/8+\bar{M}$
jmp reg	$\bar{J}$

其中,*data* 为 $\bar{N}/8$ 字节任意值.将上述 $\bar{N}/8+\bar{M}+\bar{J}$ 字节代码的功能称为 f,则 f 可表述为:控制程序流程到当前进程空间任意位置.

由于程序 P 存在可使其转移到可控区域的间断点,故当 P 运行到该间断点时,会转移到可控区域 D_i 执行上述($\bar{N}/8+\bar{M}+\bar{J}$)字节代码.因此,存在输入集 I 作用于程序 P 满足 $F=P(I)$,且$\exists f\in F$,f 功能为控制程序 P 的流程到当前进程空间任意位置,即程序 P 存在强脆弱性.证毕. □

针对 32 位 X86 体系结构,上述指令代码可以表示如下(见表 2):

Table 2 Instruction and its corresponding binary code

表 2 指令语句及其对应的二进制代码

Instruction	Binary code
mov eax, xxxxxxxx	*B8 xx xx xx xx*
jmp eax	*FF E0*

命题 2 可相应表述为:X86 体系结构下,如果 32 位常规程序 P 存在可使其转移到可控制区域 D_i 的间断点,且区域 D_i 长度不小于 7 字节,则程序 P 必然产生强脆弱性.

命题 3. 程序存在强脆弱性的充分条件 2:如果 $\bar{N}$ 位常规程序 P 存在形如 $J+N$ 的第 2 类间断点(其中,N 为可控数据),且存在不小于 $\bar{N}/8$ 字节的可控制区域 $D_i=[d_{bi},d_{ei}]$,则 P 必然存在强脆弱性.

证明:由于 N 为可控数据,D_i 为可控区域,因此可构造输入集 I,使其满足:

(1) $N=d_{bi}$.

(2) 在可控区域起始位置存放 x,即$[d_{bi}]=x$,其中,x 为 $\bar{N}/8$ 字节任意值.

因此,程序 P 在执行点 $J+N$ 运行后下一指令地址为$[N]=[d_{bi}]=x$.如果将程序执行点 $J+N$ 的功能称为 f,则由于 x 可为 $\bar{N}/8$ 字节任意值,故 f 可表述为:控制程序流程到当前进程空间任意位置.因此,存在输入集 I 作用于程序 P

满足 $F=P(I)$,且$\exists f\in F$,f 功能为控制程序 P 的流程到当前进程空间任意位置,即程序 P 存在强脆弱性.证毕. □

命题 4. 程序存在强脆弱性的充分条件 3:在 $\bar{N}$ 位常规程序 P 中,如果可以向任意地址空间写入任意 $\bar{N}/8$ 字节数据,则 P 必然产生强脆弱性.

证明:假设通过构造输入集 I 可使程序 P 向进程空间 M 处写入 K,即$[M]=K$,其中,M,K 均为任意 $\bar{N}/8$ 字节值.由于 P 为 $\bar{N}$ 位常规程序,根据定义,P 产生异常时必然调用函数 *KiUserExceptionDispatcher*,且该函数不属于模块 P,因此,*KiUserExceptionDispatcher* 存在于其他模块,且程序 P 通过输入表对其进行调用.

如果程序 P 主动退出前产生异常,则构造输入集 I,使得 M 为 P 输入表中函数 *KiUserExceptionDispatcher* 对应内存地址的存储位置,故$[M]=K$ 表示将 P 程序输入表中函数 *KiUserExceptionDispatcher* 所在内存地址修改为 K.当程序产生异常时,必然会根据输入表中 *KiUserExceptionDispatcher* 对应的函数地址进行调用,由于该函数地址被修改为任意值 K,因此可控制程序流程到当前进程空间任意位置.如果将对 *KiUserExceptionDispatcher* 的函数调用称为功能 f,则由于 K 为任意 $\bar{N}/8$ 字节值,故 f 可表述为:控制程序流程到当前进程空间任意位置.因此,存在输入集 I 作用于程序 P 满足 $F=P(I)$,且存在 $f\in F$,f 功能为控制程序 P 的流程到当前进程空间任意位置,即程序 P 存在强脆弱性.

如果程序 P 主动退出前不产生异常,假设程序调用函数 *ExitProcess* 结束程序(调用其他函数结束程序原理相同),可按照上述方法构造程序 P 的输入表中函数 *ExitProcess* 对应函数地址的存储位置,将 *ExitProcess* 对应函数地址的存储空间修改为 K,同理可证程序 P 必然存在强脆弱性.

所以,无论程序 P 主动退出前是否产生异常,P 均存在强脆弱性.证毕. □

当然,很多情况下我们并不需要修改当前进程的 *KiUserExceptionDispatcher* 或 *ExitProcess* 输入表项,即可说明程序 P 存在强脆弱性,例如可修改程序返回地址、函数 *free*()对应的输入表项或修改 SEH 链表等,而且这种修改在实际应用中更加广泛一些.其实,命题 4 可作如下更广义表述:在 $\bar{N}$ 位常规程序 P 中,如果可以向进程空间敏感位置写入任意 $\bar{N}/8$ 字节数据,则 P 必然存在强脆弱性.上述敏感位置可以为栈帧返回地址、SHE 链表中保存的异常处理函数句柄、进程输入表项或后续代码中存在 $J+N$ 型第 2 类间断点时 N 指向的内存区域等.另外,虽然有些情况下仅向任意位置写入小于 $\bar{N}/8$ 个字节的数据也可以任意控制程序流程,但这种做法不具备普遍性,因此不是存在强脆弱性的充分条件.

可以利用上述产生强脆弱性的必要条件和充分进行强脆弱性的检测,但是值得注意的是,并不存在一个万能的强脆弱性检测程序可以检测任意程序是否存在强脆弱性,下面给予证明.

命题 5. 不存在一个强脆弱性检测程序可以检测所有 $\bar{N}$ 位常规程序是否存在强脆弱性.

证明:假设存在强脆弱性检测程序 $P(X,Y)$可以检测所有 $\bar{N}$ 位常规程序是否存在强脆弱性,其中,X 表示被检测程序,Y 表示 X 的输入,那么我们可以根据 P 设计一个新的 $\bar{N}$ 位常规程序 Q,如下:

```
Program Q(X,Z)
{
    m=P(X,X);
    if (m==yes) return;
    if (m==no) __asm{call atoi(Z)};
}
```

这段程序通俗地讲就是:输入任何一段程序 X,调用函数 $P(X,X)$并得到返回值 m,如果 m=yes,即 P 判断程序 X 作用于它自身 X 存在强脆弱性,则程序返回;如果 m=no,即 P 判断程序 X 作用于它自身 X 不存在强脆弱性,则 Q 就执行 *call atoi*(Z).其中,Z 是用户输入任意数据,即跳转到任意区域执行.

如果假设 $Q(Q,Z)$会出现强脆弱性,那么 $P(Q,Q)$返回 yes,根据 Q 定义,Q 函数会马上返回,从而 $Q(Q,Z)$并不存在强脆弱性;如果假设 $Q(Q,Z)$不存在强脆弱性,那么 $P(Q,Q)$返回 no,根据 Q 定义,Q 函数会马上执行 *call atoi*(Z),程序会跳转到任意区域执行,因而 $Q(Q,Z)$会产生强脆弱性.

所以,无论 $Q(Q,Z)$会不会出现强脆弱性,都会产生逻辑悖论,因此不可能存在一个程序 P 可以检测所有 $\bar{N}$ 位常规程序是否存在强脆弱性[13,14].证毕. □

2 强脆弱性分析

强脆弱性分析是对程序代码进行分析,判断其是否存在强脆弱性.本节以异常分析为基础,提出了判断程序存在强脆弱性的 3 条基本依据和利用回溯分析进行强脆弱性分析的基本方法,并指出回溯分析的复杂度符合指数函数增长规律,且至少为 $O(2^h)$.

2.1 强脆弱性判定依据

当捕获一个系统异常后,程序的代码形式可作如下表示:

$$f(x)\rightarrow g(x)\rightarrow h(x),$$

其中,$f(x)$表示程序产生异常前执行的代码,$g(x)$表示程序产生异常的代码,$h(x)$表示程序产生异常后尚未执行的代码.系统之所以会在 $g(x)$处产生异常,有两个方面的原因:

(1) 直接扰动:输入数据直接作用于 $g(x)$处代码,从而产生异常.如 *mov* [*x*],0*x*0,其中,*x* 为输入数据.

(2) 间接扰动:输入数据在 $f(x)$运算中扰动了其他数据,或自身被其他数据所扰动,从而在 $g(x)$处产生异常,如下列代码:

```
mov eax, x
add ebx, eax
mov [ebx], 0x0
```

如果由于直接扰动产生异常,则需要分析异常之后的程序代码 $g(x)$和 $h(x)$是否存在强脆弱性;如果由于间接扰动产生异常,则不仅需要对 $g(x)$,$h(x)$进行分析,而且需要对 $f(x)$进行回溯分析以寻找可能产生强脆弱性的代码.无论由于直接扰动还是间接扰动,都需要对程序代码进行分析,寻找可能存在强脆弱性代码的位置.结合强脆弱性产生的充分条件,下面给出判定程序存在强脆弱性的基本依据:

依据 1. 程序存在可使程序转移到可控区域的间断点.

依据 2. 程序存在形如 *J*+*N* 的第 2 类间断点,其中,*N* 为可控数据.

依据 3. 可向程序任意地址写入任意 $\overline{N}$/8 字节数据,或可向敏感区域写入任意 $\overline{N}$/8 字节数据.所谓敏感区域包括栈帧返回地址、SHE 链表保存的异常处理函数句柄、进程输入表项或后续代码中存在 *J*+*N* 型第 2 类间断点时 *N* 指向的内存区域等.

一般说来,只要满足上述条件之一即可认定该程序存在强脆弱性.值得注意的是,与上一节强脆弱性存在的充分条件相比,其判定依据没有指明是否需要存在一小段可控区域.这并不代表不需要这一小段可控区域存在,主要是因为在进程空间寻找一小段可控内存并不困难,这一前提条件可以忽略不计.

2.2 回溯分析

回溯分析是对程序代码进行逆向分析的过程.若因对输入数据间接扰动而导致程序产生异常,则需要对程序进行回溯,分析其产生异常的原因,寻找可控数据,从而根据强脆弱性判定依据确定程序是否存在强脆弱性.

例 1:在 32 位 X86 体系结构系统中,假设存在如下代码:

```
mov ebx, x
mov eax, [edi+ebx*4]
add ecx, eax
call [ecx]
```

其中,*x* 为可控数据,*edi* 指向堆地址(这里的 *edi* 指寄存器 *edi* 所存储的数据,如果不加说明,后面均用寄存器名称代表其存储的数据).假设程序运行到最后一行代码时产生异常,下面通过回溯分析判断其是否存在强脆弱性.将对上述代码的回溯分析过程表示为一棵树,如图 1 所示.

需要注意的是,由于 *edi* 指向堆地址,一般说来堆地址不可控,所以回溯分析时可以忽略 *edi*.由图 1 可以确定 *ecx* 被 *x* 所控制,所以 *ecx* 也是可控数据,因此指令 *call* [*ecx*]满足强脆弱性判定依据 2,故上述代码存在强脆弱性.一般软件结构复杂、代码量大,对其进行完全回溯分析具有一定难度,为此给出一个在 *H* 层内进行广度优先回

溯跟踪的算法.

算法 1. 在 H 层内利用广度优先算法进行回溯分析.

算法:假设进行回溯的树为 $G(V,E)$,其中,V 表示树的顶点集合,E 表示树的边集合,$v_0 \in V$ 表示树的根节点.用 Q 表示一个队列,$EN(Q,x)$表示元素 x 进入 Q 队列,$x=OUT(Q)$表示出队列操作,且将 x 赋值为出队列元素,如果队列为空返回 $NULL$.用 h 表示当前回溯的层,v 表示当前访问的节点.$flag$ 为一个常量,且 $flag \notin V$.

(1) $h=0, v=v_0$;

(2) $EN(Q,flag)$;

(3) if ($h>=H$) goto (10);

(4) 访问 v 点;

(5) 将 v 的所有子节点 $v_1,v_2,\ldots,v_k$ 压入队列 Q:$EN(Q,v_1),EN(Q,v_2),\ldots,EN(Q,v_k)$;

(6) $x=OUT(Q)$;

(7) if ($x==NULL$) goto (10);

(8) if ($x==flag$){$EN(Q,flag)$;h++;$x=OUT(Q)$};

(9) goto (3);

(10) 退出.

算法中每个节点访问 1 次,且需要对每个节点的子节点进行访问,从而完成进队列操作,因此算法复杂度为 $O(n+e)$.其中,n 为节点数,e 为边数.

下面考虑一种比较极端、但在实际回溯中极为常见的回溯分析.如果只考虑由取值运算[u](其中,u 为变量)引起的回溯分析,且每个节点 x 只由 1 个因素 u 决定(实际回溯分析过程中常常如此,这是由 X86 指令系统的寻址方式决定的),则 x 之所以取得当前值,要么是归因于 u,要么归因于[$f(u)$](其中,$f(u)$表示对 u 的算术运算).如果以此类推,回溯分析可表示为如图 2 所示的二叉树形式.

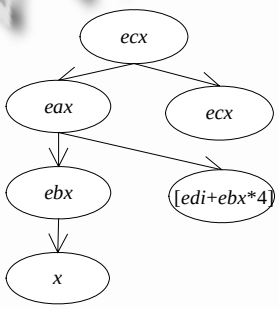

Fig.1 Backtracking process based on tree

图 1 基于树的回溯过程

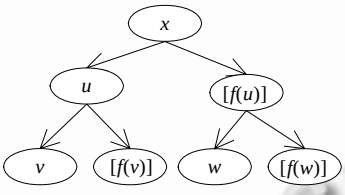

Fig.2 Backtracking process based on binary tree

图 2 基于二叉树的回溯过程

这么做的原因在于,一般的回溯分析只需对可执行代码设置断点,然后对程序进行逆向跟踪即可完成.针对取值运算的回溯分析不仅需要对可执行代码设置断点,而且往往需要设置内存断点.例如“*mov eax, [ebx]*”中,为了对 *eax* 进行回溯分析,不仅需要分析 *ebx* 何时被赋值,而且需要在 *ebx* 所指向的内存区域设置内存断点,监测该内存区域何时被写入数据.同时,取值运算在汇编代码中极为常见.因此在考虑回溯分析复杂性时,相对于取值运算,其他因素引起的回溯分析可以忽略不计.同时,在取值运算[u]中,u 为常数的情况极为少见.即使存在,跟踪起来也并不复杂.因此在计算回溯分析的复杂度时,只需要考虑取值运算[u]、且 u 为变量的情况即可.

由于目前程序的代码量很大,因此相对于人对二叉树的回溯跟踪能力来说,进行回溯分析的二叉树可以被看作一棵完全二叉树.如果算法 1 中的回溯分析基于完全二叉树,则由于 $n=2^h-1, e=2^h-2$,因此 $n+e=2^{h+1}-3$,故其算法复杂度为 $O(2^{h+1})$.

命题 6. 假定回溯分析的二叉树是完全二叉树,则异常后回溯分析的检测点相对于层的增长呈指数规律

变化,且回溯分析的时间复杂度至少为 $O(2^h)$,其中,h 为层数.

证明:完全二叉树顶点和层的关系式为 $n=2^h-1$,而回溯分析时每个顶点至少检测 1 次,因此每增加一层,异常后的回溯分析的检测点增加个数为

$$2^{h+1}-1-(2^h-1)=2^h.$$

因此,异常后的回溯分析相对层的增长符合指数函数增长规律.在对完全二叉树回溯分析时,对于二叉树的每个顶点至少分析 1 次,即其时间复杂度至少为 $O(n)$.由于 $n=2^h-1$,故回溯分析的时间复杂度至少为 $O(2^h)$.证毕. □

3 强脆弱性利用

强脆弱性利用是根据程序代码抽象出程序存在强脆弱性的具体条件,并对其进行求解的过程.本节将强脆弱性利用条件抽象成强脆弱性方程组,通过对强脆弱性方程组求解达到利用强脆弱性的目的.在对强脆弱性方程组求解中提出了相对可控、广义方程和 VC 分解等求解方法,并给出了相应算法.

3.1 强脆弱性方程组

根据强脆弱性分析的 3 条基本依据,要实现强脆弱性利用必须满足一个条件:指定某个存储空间(内存空间或寄存器)所存储的数据为某个固定值.

例 2:在 32 位 X86 体系结构系统中,假设存在如下代码:

```
        mov eax, x
        cmp eax, 8
        jb out
        mov edx, [edi+eax*4+8]
        call [edx]
out:
        ret
```

其中,x 为可控数值;*edi* 指向不稳定的堆地址,堆内存放有用户输入数据.由于 *edi* 在程序每次运行时都会临时动态分配,因此程序每次运行时 *edi* 均不相同.上述代码中,要达到任意控制程序流程的目的必须使 *edx* 指向某个可控的内存区域,为此需要解如下方程组:

$$\begin{cases}\Phi(x)=[[edi+4*x+8]]=K,\ K\text{为32位任意定值}\\ \Psi(x)=x>8\end{cases} \tag{1}$$

定义 6. 推广到一般情况,在强脆弱性利用中,最重要的就是确定并求解如下方程组:

$$\Phi(x)=K,\ K\text{为}\bar{N}\text{位任意定值} \tag{2}$$

$$A<\Psi(x)<B,\ A,B\text{为}\bar{N}\text{位定值} \tag{3}$$

由上述方程(2)、方程(3)联立的方程组称为强脆弱性方程组,称方程(2)为主方程,方程(3)为约束方程,称 $\Phi(x)$ 为强脆弱性方程组的主函数.

由程序代码准确抽象出强脆弱性方程组是强脆弱性利用的关键,强脆弱性方程组求解是强脆弱性利用的核心.大多数情况下,强脆弱性方程组并不能直接求解,其主方程求解尤为困难,需要进行变形和分解才能得到满足条件的解.下面介绍的相对可控、广义方程和 VC 分解是强脆弱性分析方程求解的常用方法.

3.2 相对可控

在方程组(1)的求解中,程序每次运行 *edi* 均不相同,但是由于堆空间往往存储有用户输入数据,因此 *edi* 某个偏移处的内存空间往往是可控区域,即$\exists x$,满足[*edi*+*x*]为可控数据,为此引入相对可控的概念.

定义 7. 多次运行同一程序,如果某个存储空间所存储的数据在程序每次运行到某个固定程序执行点时都相同,则称该存储空间所存储的数据在该程序执行点稳定;如果某个存储空间所存储的数据在程序运行到某个固定程序执行点时稳定且为可控数据,则称该存储空间所存储的数据在该程序执行点稳定可控.

定义 8. 程序运行到某执行点时,在 $x+y$ 中,如果 x 稳定可控,y 不稳定可控,造成 $x+y$ 不稳定可控,但[$x+y$]稳定可控,则称 x 相对可控.此时称 $x+y$ 为相对可控地址,称[$x+y$]为相对可控数据.

定义 9. 假设强脆弱性方程组的主方程如下:

$$\Phi(x)=h([f(x)])=K,\ x \text{ 相对可控},$$

则其对应的相对可控方程为

$$\tilde{\Phi}(x)=f(x)=N,\ N \text{ 为使 } x \text{ 满足相对可控的相对可控地址}.$$

算法 2. 假设 x 相对可控,其强脆弱性方程组的主方程为

$$\Phi(x)=h([f(x)])=K,\ K \text{ 为 } \bar{N} \text{ 位定值}.$$

利用相对可控方程求解 x 及其相对可控数据.

算法:由于 x 相对可控,联合主方程及其相对可控方程可得如下方程组:

$$\begin{cases}\Phi(x)=h([f(x)])=K,\ K\text{为}\bar{N}\text{位定值}\\ \tilde{\Phi}(x)=f(x)=N,\ N\text{为使}x\text{满足相对可控的相对可控地址}\end{cases}.$$

利用 $f(x)=N$ 求解 $x=f^{-1}(N)$.将 $f(x)=N$ 带入 $\Phi(x)=K$ 可得$[N]=h^{-1}(K)$,即

$$\begin{cases}x=f^{-1}(N)\\ [N]=h^{-1}(K)\end{cases}.$$

下面利用算法 2 对方程组(1)求解.由于两次求值运算略显复杂,因此可化简方程组(1)为下列等价方程组:

$$\begin{cases}\Phi(x)=[edi+4*x+8]=K_1,\ K_1\text{指向区域}D_i\\ \Psi(x)=x>8\\ D_i\text{稳定可控,且不小于4字节}\end{cases}\tag{4}$$

在进程空间寻找一段稳定可控内存并不难,例如,栈空间存储的数据往往稳定可控,因此确定 D_i 并不难.利用相对可控方程将方程组(4)转换为如下方程组:

$$\begin{cases}\Phi(x)=[edi+4*x+8]=K_1,\ K_1\text{指向区域}D_i\\ \tilde{\Phi}(x)=edi+4*x+8=N,\ N\text{为使}edi\text{满足相对可控的相对可控地址}\\ \Psi(x)=x>8\\ D_i\text{稳定可控,且不小于4字节}\end{cases}.$$

利用算法 2,很容易求解:

$$\begin{cases}x=(N-edi-8)/4\\ [N]=K_1,\ K_1\text{指向区域}D_i\\ D_i\text{稳定可控,且不小于4字节}\end{cases}.$$

上述解的含义是:当 x 为(N–edi–8)/4 时,内存地址 N 处所存储的数据稳定可控.若恰当构造输入集 I,使 x 为(N–edi–8)/4 且[N]为 K_1,则可稳定利用该强脆弱性.

3.3 广义方程

确定强脆弱性方程组并对其主方程求解,是强脆弱性分析的核心.直接对主方程求解往往会遇到无解或其解不能满足约束方程的情况,为此需要对主方程进行拓展.

公理 1. 在 $\bar{N}$ 位计算机系统中,如果有负数参与运算,则需要变换成补码.

公理 2. 在 $\bar{N}$ 位计算机系统中,如果参与有符号运算的 $\bar{N}$ 位数的最高位为 1,则该数可被看作负数.

定义 10. 广义方程:利用 $\bar{N}$ 位操作系统最大字长为 $\bar{N}$ 位这一特点,将 $\bar{N}$ 位值进行扩展.一般说来,主方程 $\Phi(x)=K$ 的广义方程具有如下形式:

$$\bar{\Phi}(x)=MK,\ M=0,1,2,\ldots,M_{\max},\ K \text{ 为 } \bar{N} \text{ 位定值},$$

其中,$\bar{\Phi}(x)$称为广义函数.相对于该广义方程,称主方程 $\Phi(x)=K$ 为原始方程.

例 3:令 $\bar{N}$ 取值 32,如存在方程 $A\times x=FFFFFFFC$,由于不能整除而无法直接求解.如果将方程拓展为 $A\times x=3FFFFFFFC$,则 $x=66666666$ 满足条件.方程 $A\times x=3FFFFFFFC$ 称为 $A\times x=FFFFFFFC$ 的广义方程.

命题 7. 在 $\bar{N}$ 位计算机系统中,设强脆弱性方程组的主方程及其广义方程的解集分别为 A 和 B,则 $A\subseteq B$.如果主函数及其广义函数的运算结果均存储于 $\bar{N}$ 位存储空间,则对于任意 $a\in A,b\in B$,满足 $\Phi(a)=\bar{\Phi}(b)$.

证明:强脆弱性方程组的主方程及其广义方程可分别表示为

$$\Phi(x)=MK,\ M=0,\ K \text{ 为 } \bar{N} \text{ 位定值},$$

$$\bar{\Phi}(x)=MK,\ M=0,1,2,\ldots,M_{\max},\ K \text{ 为 } \bar{N} \text{ 位定值},$$

其解集分别为 A,B,显然存在 $A\subseteq B$.

设$\exists a,b$ 分别满足 $a\in A,b\in B$,则

$$\Phi(a)=K,$$

$$\bar{\Phi}(b)=MK,\ M=0,1,2,\ldots,M_{\max}.$$

程序运行时,K 和 MK 分别被存储在某个 $\bar{N}$ 位存储空间,由于当前系统为 $\bar{N}$ 位系统,所以多于 $\bar{N}$ 位的 M 必然会被丢弃,因此 $\Phi(a)=\bar{\Phi}(b)$.证毕. □

命题 7 表明,原始方程推广到广义方程以后虽然解集扩大了,但是对程序运行并没有影响,这是广义方程存在的基础.

算法 3. 已知强脆弱性方程组如下:

$$\begin{cases}\Phi(x)=K,\ K\text{为}\bar{N}\text{位定值}\\ A<\Psi(x)<B,\ A,B\text{为}\bar{N}\text{位定值}\end{cases}.$$

利用广义方程对上述强脆弱性方程组求解.

算法:首先将上述强脆弱性方程组的主方程拓展为广义方程:

$$\bar{\Phi}(x)=MK,M=0,1,2,\ldots,M_{\max}.$$

假设 $x=x_M$ 时 $\bar{\Phi}(x)$取的最大值,则 $M_{\max}=\bar{\Phi}(x_M)/10^{\bar{N}/4}-1$(取 $\bar{\Phi}(x_M)/10^{\bar{N}/4}$ 的整数部分,如果不加说明,下面涉及到的除法运算均为此意).对上述广义方程求解

$$x_0=\bar{\Phi}^{-1}(K);$$
$$x_1=\bar{\Phi}^{-1}(1K);$$
$$\ldots$$
$$x_M=\bar{\Phi}^{-1}(M_{\max}K);$$

将 $x_0,x_1,\ldots,x_M$ 逐个带入约束方程 $A<\Psi(x)<B$,判断其是否满足约束条件,舍去不满足约束条件的解.

在强脆弱性分析中经常会需要求解广义方程,尤其是在原始方程无解或解不能满足约束方程的情况下必须对原始方程进行拓展.

例 4:在 32 位 X86 体系结构系统中,假设存在如下代码:

```
    mov eax, x
    mov ebx, y
    cmp eax, 8
    jge out
    imul eax, eax, 0AH
    mov [edi+eax], ebx
    mov ecx, [edi−12]
    call [ecx]
out:
    ret
```

其中,x,y 稳定可控,edi 指向堆地址.根据强脆弱性利用的判定依据,其强脆弱性方程组如下:

$$\begin{cases}\Phi(x)=x\times A=-12\\ x<8\end{cases}.$$

根据公理 1,其主方程需要改写为如下形式:

$$\Phi(x)=x\times A=FFFFFFEE.$$

主方程不能直接求解(不能整除),因此必须将主方程拓展为广义方程并利用算法 3 求解:

$$\bar{\Phi}(x)=x\times A=MFFFFFFEE, M=0,1,2,...,M_{\max}.$$

令 $x=FFFFFFFF$,可由 $M_{\max}=x*A/100000000-1$ 求得 M 最大值为 8.

对 M 从 0 到 8 进行测试,其广义方程有解:

M=2 时,$x_1=4CCCCCCB$,

M=7 时,$x_2=CCCCCCCB$,

根据公理 2,x_2 满足约束方程.

因此,上述代码存在强脆弱性,且当 x 为 $CCCCCCCB$ 时可稳定利用.

3.4 VC分解

定义 11. VC 分解(variable constant decompose):指将函数分解为至少含有 1 个稳定可控常量的形式.一般说来,$f(x)$分解后具有如下形式:

$$f(x)=g(x)\odot h(x),$$

其中,$\odot$为可逆运算符(即存在逆运算,如加法、循环右移等).当 x 取某个值时,$g(x)$和 $h(x)$满足至少有 1 个函数稳定可控且可取任意值.假设 $g(x)$稳定可控且可取任意值,$h(x)$不稳定可控,则 $g(x)$称为 C 函数(或常量函数),$h(x)$称为 V 函数(或变量函数).

算法 4. 已知强脆弱性方程组的主方程为 $f(x)=K$,$f(x)$可分解为 $g(x)\odot h(x)$,其中,$g(x)$为常量函数,$h(x)$为变量函数,$\odot$是可逆运算符.当 $x=x^*$时 $g(x)$满足稳定可控且可取任意值,求当 $g(x)$为何值时满足 $f(x)=K$.

算法:假设 $x=x^*$时 $h(x^*)$取值为 K_h,即 $h(x^*)=K_h$.

由于 $f(x)=K$,且 $f(x)=g(x)\odot h(x)$,所以有

$$g(x)\odot h(x)=g(x)\odot K_h=K.$$

由于$\odot$是可逆运算,如果用$\odot^{-1}$表示其逆运算符,则 $g(x)=K\odot^{-1}K_h$.

在强脆弱性方程组求解中,直接求解往往很困难,甚至无解,将其进行 VC 分解可以使问题简单化.

例 5:在 32 位 X86 体系结构系统中,假设存在如下代码:

```
mov eax, x
mov ebx, [edi+eax]
imul eax, eax, 4
add ebx, eax
call [ebx]
```

其中,x 稳定可控,edi 指向堆地址且相对可控.根据强脆弱性利用的依据,其强脆弱性方程组如下:

$$\begin{cases}\Phi(x)=[edi+x]+4*x=K,\ K\text{为32位任意定值}\\ \Psi(x)\text{任意}\end{cases}.$$

由于 $edi+x$ 相对可控,因此可进行 VC 分解,主方程分解后形式如下:

$$\begin{cases}\Phi(x)=f(x)+g(x)=K,\ K\text{ 为 32 位任意定值} & (5)\\ f(x)=[edi+x] & (6)\\ g(x)=4\times x & (7)\\ \tilde{\Phi}=edi+x=N,\ N\text{ 为使 }edi\text{ 满足相对可控的相对可控地址} & (8)\end{cases}$$

首先对相对可控方程(8)求解:

$$x=N-edi.$$

可利用算法 4 对 C 函数 $f(x)$求解.将 x 带入方程(7)可确定 $g(x)$:

$$g(x)=4\times(N-edi).$$

将 $g(x)$值带入方程(5)可确定 $f(x)$:

$$f(x)=K-g(x)=K-4\times(N-edi).$$

因此,恰当构造输入集 I,使 x 为 $N-edi$,且内存地址 N 处存储数据为 $K-4\times(N-edi)$,则可稳定利用该强脆弱性.

4 实例分析

回溯分析、相对可控、广义方程和 VC 分解是强脆弱性分析与利用中常用的分析和求解方法,下面利用这些方法对 OFFICE 系列软件中的两个异常进行分析.

4.1 实例1

在 X86 体系结构系统中,32 位常规程序 WINWORD.EXE 中存在如下代码:

```
3001bd443    mov eax, [eax+8]
3001bd446    mov ecx, [eax]
3001bd448    push eax
3001bd449    call [ecx+0x14]
```

经过测试可发现异常产生于最后一条语句,首先需要进行回溯分析,其二叉树如图 3 所示.分析后可以发现,[*eax*+8]中 *eax* 稳定可控.

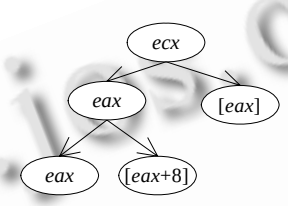

Fig.3 Backtracking analysis

图 3 回溯分析

将语句改写如下:

```
mov eax, x
mov eax, [eax+8]
mov ecx, [eax]
push eax
call [ecx+0x14]
```

可得到强脆弱性方程组:

$$\begin{cases}\Phi(x)=[[x+8]]+14=K,\ K为\overline{N}位任意定值\\ \Psi(x)任意\end{cases},$$

该方程等价于

$$\begin{cases}\Phi(x)=x=K_1-8,\ K_1指向D_i\\ \Psi(x)任意\\ D_i稳定可控,且不小于8字节\end{cases}.$$

上述条件很容易满足.

4.2 实例2

在 X86 体系结构系统中,32 位常规程序 POWERPNT.EXE 中存在强脆弱性代码,将冗余代码删除、部分代码用等价语句替换后如下所示:

```
            mov     eax, x
3002cd3c    lea     esi, [edi+eax*4]
```

```
3002cd3f   cmp     [esi], 8
3002cd42   jge     PowerPoint+0x2cd84 (3002cd84)
3002cd44   push    ebx
3002cd45   call    PowerPoint+0x2cd89 (3002cd89)
3002cd4a   test    al, al
3002cd4c   jne     PowerPoint+0x2cd84 (3002cd84)
3002cd4e   mov     eax, [ebx]
3002cd50   mov     ecx, [esi]
3002cd52   imul    eax, eax, 0Ah
           mov     [ebx+4], y
3002cd55   mov     edx, [ebx+4]
3002cd58   add     ecx, eax
3002cd5a   mov     [edi+ecx*4+8], edx
....
           mov     esi, [edi–40]
...
3002385b   mov     eax, [esi]
3002385d   mov     ecx, esi
3002385f   call    [eax]
```

其中,*x*,*y* 稳定可控,*edi* 指向堆地址且相对可控.根据上述代码,只要能够向 *edi*–40 写入任意 4 个字节就可稳定利用该强脆弱性,因此只需 *edi*+*ecx**4+8=*edi*–40,即 *ecx**4+8=–40 即可.

其强脆弱性方程组如下:

$$\begin{cases}\Phi(x)=([edi+4\times x]+x\times A)\times 4+8=-40\\ \Psi(x)=[edi+4\times x]<8\end{cases}.$$

下面分别用两种方法对其进行求解.

方法 1. 由于 *edi* 相对可控,因此[*edi*+4**x*]可以作为相对可控值出现.为了简化方程,使其值取 0.根据广义方程原则、负数变补码原则,上述方程组的可表示为

$$\begin{cases}\bar{\Phi}(x)=x\times A\times 4+8=MFFFFFFC0,\ M=0,1,2,...,M_{\max}\\ \Psi(x)=[edi+4\times x]=0\end{cases}.$$

可利用算法 3 对上述广义方程求解.

令 *x*=*FFFFFFFF*,可求得 *M* 最大值:

$$M_{\max}=(FFFFFFFF*A\times 4+8)/100000000-1=26.$$

将 *M* 从 0 到 26 逐个取值,对 *x* 求解,将 *x* 带入约束方程验证是否满足条件.

方法 2. 由于直接求解上述方程有一定难度,首先对其主方程进行 VC 分解,结合相对可控方程,分解后的方程组如下:

$$\begin{cases}\bar{\Phi}(x)=([edi+4\times x]+x\times A)\times 4+8=MFFFFFFC0,\ M=0,1,2,...,M_{\max}\\ \Psi(x)=[edi+4\times x]<8\\ f(x)=[edi+4\times x]\times 4\\ g(x)=x\times A\times 4+8\\ \tilde{\Phi}=edi+4\times x=N,\ N\text{为使}edi\text{满足相对可控的相对可控地址}\end{cases}.$$

求解相对可控方程 $\tilde{\Phi}=N$ 可得:

$$x=(N-edi)/4.$$

利用算法 4 对 *f*(*x*)求解得:

$$f(x)=[edi+4\times x]\times 4=MFFFFFFC0-8-x\times A\times 4=MFFFFFFC0-8-(N-edi)\times A.$$

因此有

$$\Psi(x)=f(x)/4=(MFFFFFFC0-8)/4-A\times(N-edi)/4=(MFFFFFFC0-A\times N+A\times edi)/4-2.$$

可调整 M,使 $\Psi(x)$最高位为 1.根据公理 2,最高位为 1 可看作负数,故可满足约束方程.

原始的强脆弱性方程组中的主方程是无解的,第 1 种解法充分利用广义方程,将主方程变换成广义方程求解,虽然可以求解,但解法略显复杂;第 2 种解法充分利用 VC 分解和公理 2,使问题得以简化.

5 结束语

本文介绍了异常捕获的基本方法,剖析了强脆弱性本质,提出了程序存在强脆弱性的必要条件和充分条件.在强脆弱性分析中,提出了判断强脆弱性的 3 条基本依据和基于二叉树的回溯分析方法.在强脆弱性利用中,将强脆弱性利用条件抽象成强脆弱性方程组,通过对强脆弱性方程组求解达到利用强脆弱性的目的.强脆弱性方程组的抽象是强脆弱性利用的关键,主方程的求解是强脆弱性方程组求解的核心.对于主方程的求解,本文提出了基于相对可控方程、广义方程和 VC 分解的求解方法,这些方法对求解主方程很有帮助.

基于异常的强脆弱性分析依然面临着很多困难,比如,由于软件结构越来越庞大,回溯分析越来越复杂,从汇编代码中准确抽象出强脆弱性方程组也变得很困难,强脆弱性方程组的主方程也越来越难以求解,尤其是存在嵌套求值运算时难以很好地求解.回溯分析、强脆弱性方程组的抽象和求解依然面临很大的挑战.

References:

[1] Sezer EC, Ning P, Kil C. MemSherlock: An automated debugger for unknown memory corruption vulnerabilities. In: di Vimercati SDC, Syverson P, Evans D, eds. Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007. 562−572.

[2] Necula G, McPeak S, Weimer W. CCured: Type-Safe retrofitting of legacy code. In: John L, ed. Proc. of the 29th ACM SIGPLANSIGACT Symp. on Principles of Programming Languages. New York: ACM, 2002. 128−139.

[3] Necula GC, Condit J, Harren M, McPeak S, Weimer W. CCured: Type-Safe retrofitting of legacy software. ACM Trans. on Programming Languages and Systems (TOPLAS), 2005,27(3):477−526. [doi: 10.1145/1065887.1065892]

[4] Castro M, Costa M, Martin JP. Better bug reporting with better privacy. ACM SIGARCH Computer Architecture News, 2008, 36(1):319−328. [doi: 10.1145/1353534.1346322]

[5] Miller PP, Cooksey G, Moore F. An empirical study of the robustness of MacOS applications using random testing. ACM SIGOPS Operating Systems Review, 2007,40(1):78−86.

[6] Forrester JE, Miller BP. An empirical study of the robustness of windows NT applications using random testing. In: Chen JB, Draves R, eds. Proc. of the 4th USENIX Windows Systems Symp. Washington: USENIX Press, 2000. 59−68.

[7] Xu QJ, Xue Z. Analysis and study of buffer overflow attack detection technology. Computer Engineering, 2007,33(16):142−152 (in Chinese with English abstract).

[8] Mitra P, Murthy CA, Pal SK. Unsupervised feature selection using feature similarity. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2002,24(4):301−312. [doi: 10.1109/34.990133]

[9] Wang Q. 0day Security: Software Vulnerability Analysis Technique. Beijing: Publishing House of Electronics Industry, 2008. 179−199 (in Chinese).

[10] Duan G. Encrypt and Decrypt. 3rd ed., Beijing: Publishing House of Electronics Industry, 2008. 306−316 (in Chinese).

[11] Russinovich ME, Solomon DA, Wrote; Pan AM, Trans. Microsoft Windows Internals (4th ed.). Beijing: Publishing House of Electronics Industry, 2007. 85−124 (in Chinese).

[12] Richter J, Wrote; Wang JH, Zhang HS, Hou LK, *et al*., Trans. Programming Applications for Microsoft Windows (4th ed.). Beijing: China Machine Press, 2006. 565−623 (in Chinese).

[13] Sipser M, Wrote; Tang CJ, Chen P, Xiang Y, Liu QH, Trans. Introduction to the Theory of Computation (2nd ed.). Beijing: China Machine Press, 2006. 108−114 (in Chinese).

[14] Hopcroft JE, Motwani R, Ullman JD, Wrote; Liu T, Jiang H, Wang HP, Trans. Introduction to Automata Theory, Languages, and Computation (2nd ed.). Beijing: China Machine Press, CITIC Publishing House, 2004. 215−221 (in Chinese).

附中文参考文献:

[7] 徐启杰,薛质.缓冲区溢出攻击检测技术的分析和研究.计算机工程,2007,33(16):142−152.

[9] 王清.0day 安全:软件漏洞分析技术.北京:电子工业出版社,2008.179−199.

[10] 段钢.加密与解密.第 3 版,北京:电子工业出版社,2008.306−316.

[11] Russinovich ME, Solomon DA,著;潘爱民,译.深入解析 Windows 操作系统(第 4 版).北京:电子工业出版社,2007.85−124.

[12] Richter J,著;王建华,张焕生,侯丽坤,等,译.Windows 核心编程(第 4 版).北京:机械工业出版社,2006.565−623.

[13] Sipser M,著;唐常杰,陈鹏,向勇,刘齐宏,译.计算机理论导引(第 2 版).北京:机械工业出版社,2006.108−114.

[14] Hopcroft JE, Motwani R, Ullman JD,著;刘田,姜晖,王捍贫,译.自动计理论、语言和计算导论(第 2 版).北京:机械工业出版社,中信出版社,2004.215−221.

时云峰(1978－),男,河北赵县人,工程师,主要研究领域为信息安全.

张金祥(1969－),男,博士,高级工程师,主要研究领域为信息安全,网络测量.

冯建华(1967－),男,博士,教授,博士生导师,主要研究领域为数据库,XML 数据库,异构数据的关键字检索,基于闪存的数据库系统.