relop→<|>|=|≥|≤|≠

这里的 *relexp* 表示一个经过了规范化的关系表达式,这些关系表达式通过二元逻辑操作符 *lop* 连接起来,或者通过一元逻辑操作符¬取反.在规范化关系表达式的内部,由关系运算符 *relop* 连接算术表达式和 0 构成.当实际程序中出现右部不为 0 的关系表达式时(例如 $f_1<f_2$),本文会将其规范化成右部为 0 的关系表达式($f_1-f_2<0.0$)来进一步分析使条件约束发生真假值转换的前提.

对于待测软件中的每一个条件约束 *cond*,面向控制流变化的攻击流程分析首先扫描其逻辑操作符 *lop* 构建的逻辑关系,即得到条件约束中哪些关系表达式取值的变化会影响整体条件约束的取值变化.然后,针对每一个关系表达式 *relexp*,讨论其取值发生变化的误差范围.例如,对于 $f_1-f_2<0.0$ && $f_2-f_3>0.0$ 这样的条件约束,我们重点讨论怎样的浮点误差会使 f_1-f_2 以及 f_2-f_3 的精确值和浮点结果会产生不同的正负号.我们称这一状况为约束 $f_1-f_2<0.0$(或 $f_2-f_3>0.0$)在精确实数计算条件下和浮点数值计算条件下真值相反.

对于软件中的某一个数值 *f*,我们将不考虑误差时 *f* 的精确实数值记为 *REAL*(*f*),而将程序经过浮点计算获得的浮点值记为 *FP*(*f*),则我们有:

$$REAL(f)=FP(f)+round(f) \tag{1}$$

这里,*round*(*f*)为数值 *f* 在程序中的总体舍入误差.许多已有的数值分析以及稳定性分析方法可以帮助我们判断舍入误差的范围[11,21,22].特别地,当数值变量符号式提取算法已经获得了 *f* 的计算表达式时,本文方法通过仿射分析[22]来获得(affine analysis)对应的误差表达式 *round*(*f*).

对于经过规范化的关系表达式 *relexp*,我们不失一般性地讨论其中的 3 种关系运算符>,<和=;而对于另外 3 种关系运算符≥,≤和≠,可以通过逻辑操作符合我们讨论的 3 种关系运算符复合而获得.例如,*f*≠0 可以写成¬*f*=0 的形式.下面我们讨论如何找到使精确实数计算条件下和浮点数值计算条件下真值相反的关系表达式.

定理 1. 当且仅当式(2)、式(3)成立时,关系表达式 *f relop* 0 的真值结果会因为数值误差而发生变化,即 *FP*(*f*) *relop* 0 和 *REAL*(*f*) *relop* 0 的真值相反:

$$|round(f)|>|FP(f)| \tag{2}$$

$$FP(f)\times round(f)\leqslant 0 \tag{3}$$

证明:先证明充分性.

当|*round*(*f*)|>|*FP*(*f*)|和 *FP*(*f*)×*round*(*f*)<0 成立时,首先 *FP*(*f*) *relop* 0 必然存在真值 *T* 或者 *F*;又因为 *REAL*(*f*)=*FP*(*f*)+*round*(*f*),由条件(2)、条件(3)可知,此时如果 *FP*(*f*)为正值或负值,那么 *REAL*(*f*)必然变号,所以真值就发生了翻转;如果此时 *FP*(*f*)为 0,那么 *REAL*(*f*)=*round*(*f*)>0 也必然不等于 0,此时的真值也发生了翻转.得证.

再证明必要性.

当简单判断语句的真值结果发生实数域和浮点域上的翻转时,我们也分两种情况讨论.

- 当 *FP*(*f*)不等于 0 时,不妨设 *FP*(*f*)为正值,因为 *REAL*(*f*)=*FP*(*f*)+*round*(*f*),要使 *REAL*(*f*)的值变成负值,必然要使|*round*(*f*)|>|*FP*(*f*)|且 *round*(*f*)为负值,所以条件(2)、条件(3)成立;
- 当 *FP*(*f*)等于 0 时,要使 *REAL*(*f*)的值不等于 0,必然要求|*round*(*f*)|>0,此时条件(2)、条件(3)也成立.

所以,定理 1 得证. □

本文控制流分析的目的在于找到对应的攻击输入使精确实数计算条件下和浮点数值计算条件下关系表达式的真值相反.鉴于目前 *f* 和 *round*(*f*)的符号式已知,按照定理 1 的要求,我们通过构建约束来获得攻击输入.所构建的攻击约束 *attcond* 为

$$|round(f)|-f>0\wedge round(f)\times f\leqslant 0 \tag{4}$$

当攻击约束 *attcond* 在约束求解器中可解时,我们通过约束求解直接获得对应的攻击输入值,而当软件逻辑较为复杂,使攻击约束 *attcond* 中存在非线性分量时,我们通过随机搜索方式搜索攻击约束的可满足解,从而获得满足条件的攻击输入.

2.2.2 数据依赖变化分析

除了控制流程的改变以外,数据依赖的变化也同样会造成软件存在安全漏洞问题.由于数据依赖变化的实

质是黑客访问权限的变化,例如,原先并不依赖于环境变量的密码值等敏感信息,由于数值稳定性问题而产生了依赖关联,并使得黑客有权限能够访问到这一部分信息.因此,数值误差造成数据依赖变化的漏洞隐患会对于软件产生重大威胁.

从理论上说,污点分析(taint analysis)方法[23,24]被称为信息流追踪技术,是处理和分析数据依赖,防止数据完整性和保密性被破坏的有效手段.该技术通过对系统中敏感数据进行标记,继而跟踪标记数据在程序中的传播,以检测系统安全问题.例如,当我们把软件输入等不被信任的数据源看作“被污染”的数据时,污点分析可以获得“被污染”数据的波及范围,即黑客可能访问或修改的数据范围.然而,传统的污点分析方法一般未考虑到数值误差对“被污染”数据的影响,即由于数值误差的积累可能引起“被污染”数据的波及范围发生变化,即本文所述的数值稳定性相关数据依赖变化.因此,面向数据依赖变化攻击流程分析可以采用引入误差计算的污点分析和不引入误差计算的污点分析获得各自的“被污染”数据范围,再通过比较这两个数据范围的差异来分析攻击输入.然而,经过我们的实践证明,这样的分析方式效率太低.结合精度分析的污点分析需要消耗大量的计算资源,因此,本文在实现上采用了针对当前计算机体系结构的优化分析方案来解决检测由数值误差引起的数据依赖变化及其相关的漏洞隐患.

当前主流的计算机体系结构中,一般用整数值来标定内存地址,因此,现代程序设计语言的申请内存、分配内存、释放内存等等存储操作一般也都用整数值来标定其大小和偏移.整数类型是程序中数据传播不可或缺的重要标识.我们发现:软件的浮点数值误差一般都要通过影响整数类型值从而进一步影响到数据依赖,而使黑客能够进一步改变数据的访问权限.否则,软件执行过程中“被污染”的数据范围不会因为误差的引入而发生变化.例如,当浮点数值被转换成某一整型的指针时,由于误差而使得指针位置发生变化,从而使软件的数据依赖发生变化.因此,软件中浮点数值向整数进行传播是本文分析数据依赖变化的关键点.而当该传播可能引起传播后整数值发生变化时,我们即可构建相应的攻击输入,以便在后续步骤中进一步验证是否会真实存在安全漏洞隐患.

我们重点分析软件中浮点数值向整数值的传播.本文分析方法首先标定软件中各变量的数据依赖,并以此来搜索依赖于浮点数值的整数变量 i,与此同时,我们通过与算法 1 类似的方法获得变量 i 的数值变量符号表达式 g 以及 $round(g)$的符号式.最终,我们通过构建约束来获得攻击输入.所构建的攻击约束 *attcond* 为

$$|round(g)|>0.5 \tag{5}$$

同理,当攻击约束 *attcond* 在约束求解器中可解时,我们直接解得攻击的输入值;当攻击约束 *attcond* 不可直接求解时,我们通过随机搜索方式搜索获得满足条件的攻击输入.

2.3 高精度动态攻击验证

由于静态分析本身的技术限制,其输出可能会存在误报(false positive)问题.我们可以通过高精度动态验证的方式来消除误报,获得最终的安全漏洞隐患.高精度动态验证首先将原始待测软件中的每一个浮点数值转换成运行时分配内存的高精度版本.通过大内存计算这些数值程序会极大地减少每一步的舍入误差,且由于可以在运行时动态增加每一个浮点数值的表示内存,我们可以通过以不同的内存大小多次运算来动态判断当前各变量的执行精度.当执行精度不够时,我们可以通过增加内存来动态地提升精度,获得需要的结果.因此,我们可以保证高精度数值计算在一定有效数字的范围内就是实数计算的结果.

本文通过动态比对待测试软件与对应的高精度软件版本来验证第 2 阶段分析获得的攻击输入是否有效.当我们以第 2 阶段的静态分析方法获得了软件的攻击输入之后,以该输入同时驱动待测试软件与对应的高精度软件版本执行,并在执行过程中不断比对对应的软件行为是否存在不同.如果对应的软件行为不同,则意味着误差引起了软件行为的改变,检测到的安全漏洞隐患即为真实的漏洞隐患;反之,如果因为静态分析的精度不够,对应的软件行为仍完全一致,则意味着实际运行时误差积累并未使软件行为发生变化,对应的安全漏洞隐患为误报,会在输出前被过滤.

本文基于软件剖析技术(software profiling)来比对待测软件与对应高精度软件在软件行为上的不同点.软件剖析会在当前软件源代码中的每一个条件约束与数值依赖的整数变量位置插装一条记录语句,记录当前的位置标号、条件约束的取值、整数变量的取值以及内存中的关键数据对象,并在高精度软件的相同位置插装记

录对应信息的记录语句.这里的位置标号用于追踪软件在当前输入下的执行路径,当待测软件与高精度软件在执行路径上完全一致时,我们认为软件的误差积累没有引起控制流发生变化.软件的数据依赖信息则通过在软件剖析中记录的其他信息来分析获取.

这里,我们引入了污点分析来获得软件剖析所需记录的关键内存数据对象,并以这些数据对象作为误差积累是否引起数据依赖变化的标准.通过污点推进的方式,污点分析静态定位了输入所能影响到的内存区域,对于不受输入影响的变量以及指针所指向的内存对象,在我们的检测方法中被称为保护区,本文的动态验证方法直接将与保护区相关的变量作为关键内存对象的搜索来源.对于类型为指针的相关变量,本文的软件剖析会进一步序列化指针所指向的内存对象,并将整个对象值作为数据依赖比对的依据.当待测软件与对应高精度软件的关键内存对象在相同的测试输入下存在结构上的不同(例如数组大小不同、单链表长度不同等等)或者非数值类型的变量值差异(例如 int,char,bool 等类型的数值差异)时,我们判定软件的数据依赖因为数值误差的引入而发生的变化.所检测到的安全漏洞隐患也即得到确认.

3 实例研究

本文在 9 个著名的开源项目上进行了实验,分析代码中是否存在安全漏洞隐患,通过这些实例研究,我们希望能够回答以下 3 个重要的研究问题.

- 研究问题 1:数值稳定性相关的安全漏洞隐患在软件中是否普遍存在?
- 研究问题 2:本文提出的动静态相结合的检测方法是否适用于大型软件的漏洞隐患检测?
- 研究问题 3:数值误差要积累到怎样的程度才会造成软件中存在稳定性相关的安全漏洞隐患?

3.1 实验设置

我们实现了数值稳定性相关安全漏洞隐患检测方法的工具原型,其中,数值变量符号式提取模块是基于 KLEE 1.3.0 开发的,静态攻击流程分析基于 ROSE compiler 0.9.5a 开发,动态验证模块基于 iRRAM 2013.01 开发[17].实验对象以 gcc 5.4.0 编译,并通过 Python2.7.10 来驱动测试.整体的测试环境运行在一台 CPU 为 Intel i5-5257U 2.7GHz*2 的双核处理器,内存 8GB 的笔记本计算机上,操作系统为 Ubuntu14.0.1.

我们以 9 个业界广泛使用的开源项目作为实验对象,其具体情况见表 1.为了检验本文检测方法的有效性,降低因实验对象的偶然因素带来的偏差,我们尽量选择不同类别的大规模开源项目的最新版本.其中,最大实验对象的代码行数超过了 700 万行,软件包大小为 1.42GB;最小的实验对象也达到了 50 万行的代码规模,软件包大小为 36.7MB.

Table 1 Details of our experimental objects

表 1 实验对象详细信息

项目	大小	源代码文件数	源代码总行数	版本号	项目类别
Blender	189.9MB	2 752	1 745 612	2.78c	游戏引擎
Clang	121.5MB	6 821	1 396 611	3.9.1	编译器
Crystal Space	453.2MB	1 149	844 094	2.0	游戏引擎
Firefox	1.42GB	12 961	7 187 165	50.0.1	浏览器
MPlayer	109.1MB	2 746	1 275 858	1.3.0	视频播放器
Mysql	429MB	2 564	2 425 082	5.7.17	数据库
Nebula Device 2	48.6MB	1 246	576 081	vc8	游戏引擎
OpenSceneGraph	36.7MB	1 851	512 536	3.4	游戏引擎
PHP	167.1MB	1 027	1 168 792	7.1.0	网页开发语言

为了对各项目中的数值相关代码进行快速检索,提高本文方法的检测效率,在实施本文的安全漏洞检测之前,我们通过静态预处理标定了各项目中的数值代码片段.当项目源代码中存在由浮点数计算的条件判断或由浮点数参与的强制类型转换时,对应的浮点数上下文操作即被标定为一个数值代码片段.经过标定的数值代码片段集,即为本文软件行为变化的候选分析区,来实施本文三阶段的安全漏洞隐患检测.

当我们检测到实验对象中存在由数值稳定性引起的控制流路径变化时,如果对应路径条件在循环中,我们

会进一步通过后处理来检测循环相关的拒绝服务(denial-of-service,简称 DoS)安全漏洞隐患.即检测是否存在攻击输入使该循环卡死.当存在这样的攻击输入时,黑客可以反复尝试这样的攻击输入,使当前项目中各个线程均陷入这样的死循环中,直至系统资源耗尽而拒绝再为客户端提供服务为止.

当循环中存在因误差积累而引起的数值稳定性相关拒绝服务漏洞隐患时,误差积累会造成循环条件在当前输入下始终为真(或始终为假).此时,我们进一步分析攻击输入在循环迭代后是否会因为误差,而在循环条件的关系表达式中形成不动点,或者发生反向变化,即可判断是否存在循环相关的拒绝服务(denial-of-service,简称 DoS)安全漏洞隐患.这里的反向变化是指在不考虑误差时,规范化关系表达式的左部随着循环迭代的次数的增加而变小(或变大),而考虑误差时对应的关系表达式左部随着循环迭代的次数的增加而变大(或变小).

3.2 实验结果

表 2 列出了本文检测方法在 9 个开源项目上的实验检测结果,我们在 9 个开源项目共 4 848 个数值代码片段中共发现 77 个数值稳定性相关的安全漏洞隐患.从表 2 的数值代码片段标定数量可以看出:Blender 中含有的数值代码最多,共有 2 056 个数值代码片段;Crystal Space 中含有的数值代码量最少,仅检测到 82 个数值代码片段.无论代码的规模多大,各实验对象或多或少地都存在一些数值稳定性相关的软件安全漏洞隐患.由此可知,数值稳定性相关的安全漏洞隐患目前在软件中是普遍存在的,且安全漏洞隐患的数量基本和数值代码片段的数量正相关,其线性相关系数为 0.903.另外,对于每一个实验对象,我们检测到因数值稳定性引起的控制流路径变化数量均高于因数值稳定性引起的数据依赖变化数量,我们检测到 77 个数值稳定性相关的安全漏洞隐患中有 62 个是控制流路径的变化,而仅有 15 个是数据依赖的变化.因此,数值稳定性引起的控制流变化相对于数据依赖变化来说更为普遍.

Table 2 Numbers of detected Suspicious vulnerabilities related to numerical stability

表 2 数值稳定性相关安全漏洞隐患的检测结果数量

项目	数值代码片段的位置标定数量	数值稳定性相关的漏洞隐患总数	数值稳定性引起数据依赖的变化数量	数值稳定性引起控制流路径的变化数量	数值稳定性引起循环相关的 DoS 漏洞隐患数量
Blender	2 056	24	5	19	7
Clang	486	1	0	1	1
Crystal Space	82	4	1	3	0
Firefox	1 070	17	4	13	4
MPlayer	471	10	2	8	0
Mysql	141	2	0	2	0
Nebula Device 2	166	8	1	7	0
OpenSceneGraph	267	8	2	6	0
PHP	109	3	0	3	2
总和	4 848	77	15	62	14

我们进一步分析了实验对象中由数值稳定性引起的循环相关 DoS 漏洞隐患.作为由数值稳定性引起控制流变化的特例,我们共在 4 848 个数值代码片段中发现 14 个由数值稳定性引起的循环相关 DoS 漏洞隐患(见表 2).表 3 给出了这 14 个数值稳定性引起的循环相关 DoS 漏洞隐患的具体情况.我们手工确认了每一个漏洞隐患,在表 3 中列出了利用对应漏洞的攻击输入以及攻击发生时的相对误差.另外,我们采用了文献[25]中使用的启发式数值程序测试用例生成方法,为每一个数值输入生成了 460 个测试用例来进行测试,并在表 3 中列出了观测到的最大相对误差及对应的测试输入.

由表 3 列举的数据可知,产生 DoS 漏洞隐患的数值误差往往很小,在我们的实验中,其相对误差均小于 3.00E−9,最小的情况仅需要 7.39E−42 的相对误差就可以造成数值程序存在安全漏洞.不仅如此,对应的攻击输入也往往并不在使数值误差累积到最大值的时候出现.从表 3 中我们可以看到,用例中观测到的最大相对误差往往大于漏洞被利用时的相对误差.由此可见,很小的数值误差累积就可能会造成软件中存在相关的安全漏洞隐患,传统的数值稳定性分析方法在检测相关的安全漏洞隐患时会因为仅关注较大的数值误差累积而失效.

值得注意的是,本文方法所检测到的漏洞隐患可能并不一定会被确认为当前版本的软件漏洞,但一般来说都值得软件开发人员进一步关注.例如,某软件的后台业务模块中存在因数值稳定性问题而引起的控制流变化

或者数据依赖变化,但由于前台模块的功能尚未完善,黑客在当前版本软件中并不能从已有功能点中找到攻击方案,因此并不能成为当前版本的软件漏洞.随着软件版本升级和前台模块功能的加强,在下一版本中,这一隐患就可能会演变成实际的漏洞.对于本文实验中检测到的漏洞隐患,我们正在与相关人员合作,进行进一步的确认与追踪.

Table 3 Loop related DoS vulnerability hazards

表 3 循环相关的 DoS 漏洞隐患

漏洞隐患所在项目	源代码文件名	漏洞代码段起始行号	漏洞被利用时的相对误差(攻击输入)	观测到的最大相对误差(对应输入)
Blender	key.c	662	8.13E−10(1.23E09)	1.46E−08(1.67E03)
Blender	key.c	995	1.61E−12(6.23E11)	3.82E−08(2.43E01)
Blender	shadeoutput.c	104	1.72E−14(5.82E13)	1.27E−08(2.47E05)
Blender	scaling.c	946	1.19E−13(8.41E12)	3.05E−08(7.83E01)
Blender	scaling.c	1 077	7.09E−13(1.41E12)	3.26E−08(1.35E00)
Blender	interface_draw.c	1 275	6.76E−11(1.48E08)	1.06E−08(3.47E02)
Blender	node_draw.c	687	4.05E−10(2.47E07)	1.12E−09(6.79E00)
Clang	shortest-path-suppression.c	12	2.14E−09(4.67E08)	4.27E−09(2.34E08)
Firefox	cairo-stroke-style.c	245	7.09E−13(1.41E12)	1.86E−08(5.39E07)
Firefox	audio_encoder_opus_unittest.cc	28	6.76E−21(1.48E18)	4.26E−09(2.35E06)
Firefox	pixman-conical-gradient.c	10	1.17E−19(5.39E19)	3.13E−17(8.58E01)
Firefox	pixman-conical-gradient.c	13	8.50E−22(7.39E21)	1.79E−18(1.00E13)
PHP	gd_rotate.c	518	6.36E−19(5.66E20)	4.49E−17(6.48E02)
PHP	gd_rotate.c	522	7.39E−42(4.87E43)	3.15E−17(8.87E05)

4 漏洞修复途径

本节主要讨论对于数值稳定性引发安全漏洞的修复问题.一般来说,当采用本文方法找到相关安全性漏洞后,可以考虑以下 3 种修复途径.

- 通过采用更加稳定的数值算法来修复该类漏洞.很多情况下,开发人员在软件中由于采用了不稳定的数值算法,从而使软件存在安全漏洞,这种情况下,采用更加稳定是数值算法显然是修复该类漏洞的最佳手段.例如,对于线性方程组的求解问题,采用列主元素消去法显然会比原始的高斯消去算法在稳定性上更具有优势.该修复手段通过解决数值稳定性问题,而从根源上修复该类漏洞.
- 通过解耦黑客的攻击路径,使其与相应的数值计算相互独立来修复该类漏洞.有时候,软件中的安全漏洞并不是由于开发人员采用了不稳定的数值算法而引起的,或者由于某些原因不能修改软件中的数值计算过程.此时,可以考虑将黑客可能的攻击路径与数值计算逻辑解耦,使其与软件中的数值计算过程相互独立,这样,即使软件的数值计算部分仍然存在误差,也不再会使软件存在安全性漏洞.
- 通过提高数值计算的精度来修复该类漏洞.直接提高数值计算的精度是减少数值误差的有效手段,也会成为修复数值稳定性引发安全漏洞的有效手段.然而,这一方法常常以增加计算负载,消耗更多的计算资源为代价.因此,我们建议将该方法作为修复软件安全性漏洞的备用途径,在可以通过其他方法修复漏洞时,尽量采用其他修复途径.

5 相关工作

本文给出了一种数值稳定性相关安全漏洞隐患的自动化检测方法,它能够帮助设计开发人员检测软件中由于数值稳定性而引入的安全漏洞隐患.由于目前并没有针对数值稳定性安全漏洞检测方面的研究,这里我们主要调研了数值稳定性自动化分析的相关研究[26−28].

长期以来,数值计算作为一个专门的研究领域,在探讨误差积累与结果偏差的关系、寻求尽可能减少误差积累的数值算法等方面取得了许多重要的研究成果[6−8,29,30].近年来,软件工程领域出现了一系列数值稳定性的自动化分析技术,以解决软件规模与复杂性增加、难以人工进行数值稳定性分析等问题.例如,Putot 等学者基于仿射算术法则(affine arithmetic)来分析软件可能达到的最大误差积累[22],并提出了基于这项技术的静态程序分

析[31]以及抽象解释方法[32],使分析精度大为提升;Benz 等学者给出了一套浮点数值分析与调试工具 FPDebug,该工具通过基于二进制操作的轻量级程序切片(program slicing)技术来发现软件中的数值错误[9];Bao 等学者进一步通过追踪容易出错的浮点数值来提高数值错误的发现概率[32];Boldo 等学者从形式化验证的角度来自动分析 C 程序的浮点操作性质[19,20];Darulova 等学者针对数值稳定性问题给出了一套类型系统[33],以便通过类型系统的推理来推断实际的误差积累;Martel 提出了数值误差的前向推导语义[27],并基于这一语义开发了一个优化器[28],以便将程序优化成精度更高的算法[18];Monniaux 针对不同硬件平台与编译器条件下浮点操作在语义上的弱点给出了验证方法[34].在测试用例生成方面,Miller 等学者提出了一种称为数值最大化法则的搜索技术[35],并以此来制导测试用例的生成;Bagnara 等学者基于启发式搜索来求解混合执行(concolic testing)中带有浮点数值的符号约束[36];Barr 等学者针对类似于上溢出与下溢出等浮点数异常来检测符号执行中的浮点错误[10].更进一步地,Zou 等学者通过遗传算法来改进 FPDebug 的能力,并能自动找到使结果发生大量偏差的程序输入[37].许多近年来的研究工作采用随机采样方法来分析数值程序的稳定性,其中有代表性的方法称为 CESTAC[11],它通过在数值计算中引入随机进位模式来估测其误差积累.Vignes 等学者基于 CESTAC 算法提出了离散随机算术,并构建了可用的数值计算分析库 CADNA[21].Parker 等学者对蒙特卡洛算术(Monte Carlo arithmetic,简称 MCA)进行了形式化[38],并以此作为随机数值稳定性分析的理论基础.稍后,Eggert 等学者基于这一理论构建了工具 wonglediff[39],在硬件 FPU 层面对误差进行估计.本文作者也参与了数值稳定性自动分析技术的相关研究,在已有工作的基础上引入了表达式的随机变换,并以此来分析与诊断软件的数值稳定性[25,40].以上这些工作主要关注于数值计算本身的误差积累程度以及运算结果的正确性.但它们仍然未能解决数值稳定性引起的软件安全性问题,本文针对这一新问题给出了有效的解决方案.

6 结束语

安全漏洞检测是保障软件安全性的重要手段之一,在软件安全领域受到广泛关注.随着互联网的发展,黑客的攻击手段日趋多样化,且攻击技术不断翻新,使软件安全受到了新的威胁.本文描述了当前软件中实际存在的一种新类型的安全漏洞隐患,我们称其为数值稳定性相关的安全漏洞隐患.利用这种新类型的漏洞,黑客可以绕过现有的软件安全防护措施而对系统实施攻击,且已有的数值稳定性分析方法主要关注数值误差的极大值点,很难检测到误差累积较少就会引入的安全漏洞,因而这一新类型的安全漏洞隐患十分危险.

面对这一挑战,本文首先从数值稳定性引起软件行为改变的角度对数值稳定性相关的安全漏洞隐患给出了明确定义,并提出了相应的自动化检测方法.该方法基于动静态相结合的程序分析与符号执行技术,通过数值变量符号式提取、静态攻击流程分析、以及高精度动态攻击验证这 3 个步骤来检测和分析软件中可能存在的数值稳定性相关安全漏洞.我们在业界多个著名开源软件上进行了实例研究,实验结果表明:数值稳定性相关的安全漏洞隐患与实际软件中数值误差的积累程度无关,很多误差积累很小的模块也同样会存在安全漏洞隐患.正因为如此,数值稳定性相关的安全漏洞隐患在许多含有数值计算代码的实际软件中普遍存在,且安全漏洞隐患的数量同软件中数值计算的代码量正相关.另外,实验结果也说明了本文方法能够有效地检测到实际大型软件中真实存在的数值稳定性相关安全漏洞隐患.

References:

[1] Symantec. Internet Security Threat Report. 2012. https://www.symantec.com/content/dam/symantec/docs/reports/istr-17-2012-en.pdf

[2] Symantec. Internet Security Threat Report. 2017. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf

[3] Wassermann G, Su Z. Static detection of cross-site scripting vulnerabilities. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 171−180. [doi: 10.1145/1368088.1368112]

[4] Møller A, Schwarz M. Automated detection of client-state manipulation vulnerabilities. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 749−759. [doi: 10.1109/ICSE.2012.6227143]

[5] Zheng Y, Zhang X. Path sensitive static analysis of Web applications for remote code execution vulnerability detection. In: Proc. of the 2013 Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2013. 652−661. [doi: 10.1109/ICSE.2013.6606611]

[6] Higham NJ. Accuracy and Stability of Numerical Algorithms. 2nd ed., Society for Industrial and Applied Mathematics, 2002.

[7] Miller W. Software for roundoff analysis. ACM Trans. on Mathematical Software, 1975,1(2):108−128. [doi: 10.1145/355637.355639]

[8] Wilkinson JH. Rounding Errors in Algebraic Processes. Dover Publications, Incorporated, 1994.

[9] Benz F, Hildebrandt A, Hack S. A dynamic program analysis to find floating-point accuracy problems. ACM SIGPLAN Notices, 2012,47(6):453−462. [doi: 10.1145/2254064.2254118]

[10] Barr ET, Vo T, Le V, Su Z. Automatic detection of floating-point exceptions. In: Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 2013. 549−560. [doi: 10.1145/2429069.2429133]

[11] Vignes J. A stochastic arithmetic for reliable scientific computation. Mathematics and Computers in Simulation, 1993,35(3): 233−261. [doi: 10.1016/0378-4754(93)90003-D]

[12] Zhao SZ. A reliable computing algorithm and its software application ISReal for arithmetic expressions. Science China Information Sciences, 2016,46:698−713 (in Chinese with English abstract). [doi: 10.1360/N112015-00061]

[13] NVD. CVE-2010-4467. 2010. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4467

[14] NVD. CVE-2010-4645. 2011. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4645

[15] Chen G, Jin H, Zou D, Dai W. On-Demand proactive defense against memory vulnerabilities. In: Proc. of the Network and Parallel Computing. Berlin, Heidelberg: Springer-Verlag, 2013. 368−379. [doi: 10.1007/978-3-642-40820-5_31]

[16] Caballero J, Grieco G, Marron M, Nappa A. Undangle: Early detection of dangling pointers in use-after-free and double-free vulnerabilities. In: Proc. of the 2012 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2012. 133−143. [doi: 10.1145/2338965.2336769]

[17] Müller NT. The iRRAM: Exact arithmetic in C++. Lecture Notes in Computer Science, 2001,2064:222−252. [doi: 10.1007/3-540-45335-0_14]

[18] Martel M. Program transformation for numerical precision. In: Proc. of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. Savannah: ACM Press, 2009. 101−110. [doi: 10.1145/1480945.1480960]

[19] Boldo S, Filliatre JC. Formal verification of floating-point programs. In: Proc. of the 18th IEEE Symp. on Computer Arithmetic 2007 (ARITH 2007). 2007. 187−194. [doi: 10.1109/ARITH.2007.20]

[20] Boldo S, Melquiond G. Flocq: A unified library for proving floating-point algorithms in Coq. In: Proc. of the 20th IEEE Symp. on Computer Arithmetic (ARITH). 2011. 243−252. [doi: 10.1109/ARITH.2011.40]

[21] Jézéquel F, Chesneaux JM. CADNA: A library for estimating round-off error propagation. Computer Physics Communications, 2008,178(12):933−955. [doi: 10.1016/j.cpc.2008.02.003]

[22] Putot S, Goubault E, Martel M. Static analysis-based validation of floating-point computations. In: Alt R, Frommer A, Kearfott RB, Luther W, eds. Proc. of the Numerical Software with Result Verification. Berlin, Heidelberg: Springer-Verlag, 2004. 306−313. [doi: 10.1007/978-3-540-24738-8_18]

[23] Wang K, Zhang Y, Liu P. Call me back!: Attacks on system server and system apps in Android through synchronous callback. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2016. 92−103. [doi: 10.1145/2976749.2978342]

[24] Wang L, Li F, Li L, Feng XB. Principle and practice of taint analysis. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):860−882 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5190.htm [doi: 10.13328/j.cnki.jos.005190]

[25] Tang E, Barr E, Li X, Su Z. Perturbing numerical calculations for statistical analysis of floating-point program (in)stability. In: Proc. of the 19th Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2010. 131−142. [doi: 10.1145/1831708.1831724]

[26] Goubault E, Putot S. Static analysis of numerical algorithms. In: Proc. of the 13th Int'l Static Analysis Symp. 2006. 18−34. [doi: 10.1007/11823230_3]

[27] Martel M. Propagation of roundoff errors in finite precision computations: A semantics approach. In: Proc. of the Programming Languages and Systems. 2002. 159−186. [doi: 10.1007/3-540-45927-8_14]

[28] Martel M. Semantics-Based transformation of arithmetic expressions. In: Proc. of the Static Analysis. 2007. 298−314. [doi: 10.1007/978-3-540-74061-2_19]

[29] Miller W. Toward mechanical verification of properties of roundoff error propagation. In: Proc. of the 5th Annual ACM Symp. on Theory of Computing. New York: ACM Press, 1973. 50−58. [doi: 10.1145/800125.804035]

[30] Miller W, Spooner D. Software for roundoff analysis, II. ACM Trans. on Mathematical Software, 1978,4(4):369−387. [doi: 10.1145/356502.356496]

[31] Goubault E. Static analyses of the precision of floating-point operations. In: Proc. of the 8th Int'l Static Analysis Symp. 2001. 234−259. [doi: 10.1007/3-540-47764-0_14]

[32] Bao T, Zhang X. On-the-Fly detection of instability problems in floating-point program execution. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. New York: ACM Press, 2013. 817−832. [doi: 10.1145/2509136.2509526]

[33] Darulova E, Kuncak V. Trustworthy numerical computation in scala. In: Proc. of the 2011 ACM Int'l Conf. on Object Oriented Programming Systems Languages and Applications. New York: ACM Press, 2011. 325−344. [doi: 10.1145/2048066.2048094]

[34] Monniaux D. The pitfalls of verifying floating-point computations. ACM Trans. on Programming Languages and Systems, 2007, 30(3):1−41. [doi: 10.1145/1353445.1353446]

[35] Miller W, Spooner DL. Automatic generation of floating-point test data. IEEE Trans. on Software Engineering, 1976,SE-2(3): 223−226. [doi: 10.1109/TSE.1976.233818]

[36] Bagnara R, Carlier M, Gori R, Gotlieb A. Symbolic path-oriented test data generation for floating-point programs. In: Proc. of the IEEE 6th Int'l Conf. on Software Testing, Verification and Validation (ICST). 2013. 1−10. [doi: 10.1109/ICST.2013.17]

[37] Zou D, Wang R, Xiong Y, Zhang L, Su Z, Mei H. A genetic algorithm for detecting significant floating-point inaccuracies. In: Proc. of the 37th Int'l Conf. on Software Engineering. 2015. 20−22. [doi: 10.1109/ICSE.2015.70]

[38] Parker DS, Pierce B, Eggert PR. Monte Carlo arithmetic: How to gamble with floating point and win. Computing in Science & Engineering, 2000,2(4):58−68. [doi: 10.1109/5992.852391]

[39] Eggert PR, Parker DS. Perturbing and evaluating numerical programs without recompilation: The wonglediff way. Software Practice and Experience, 2005,35(4):313−322. [doi: 10.1002/spe.637]

[40] Tang E, Zhang X, Muller N, Chen Z, Li X. Software numerical instability detection and diagnosis by combining stochastic and infinite-precision testing. IEEE Tran. on Software Engineering, 2017,PP(99):975−994. [doi: 10.1109/TSE.2016.2642956]

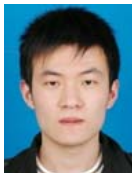
附中文参考文献:

[12] 赵世忠.算术表达式的一种可信计算算法及其软件 ISReal.中国科学:信息科学,2016,46:698−713. [doi: 10.1360/N112015-00061]

[24] 王蕾,李丰,李炼,冯晓兵.污点分析技术的原理和实践应用.软件学报,2017,28(4):860−882. http://www.jos.org.cn/1000-9825/5190.htm [doi: 10.13328/j.cnki.jos.005190]

沈维军(1989－),男,江苏盐城人,博士生,CCF 学生会员,主要研究领域为软件测试,软件数值分析.

陈鑫(1975－),男,博士,副教授,主要研究领域为软件工程,软件测试,验证技术.



汤恩义(1982－),男,博士,助理研究员,CCF 专业会员,主要研究领域为软件工程,新型软件测试方法,程序分析方法.

李彬(1988－),男,学士,主要研究领域为软件工程,程序分析,程序验证.

陈振宇(1978－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试,软件分析,智能软件工程.

翟娟(1988－),女,博士,助理研究员,CCF 专业会员,主要研究领域为程序分析与验证,程序合成.