基于黑盒遗传算法的 Android 驱动漏洞挖掘

何 远[1),2)] 张玉清[1)] 张光华[3)]

[1)](中国科学院大学国家计算机网络入侵防范中心 北京 101408)

[2)](大理大学数学与计算机学院 云南 大理 671003)

[3)](中国科学院信息工程研究所 物联网信息安全技术北京市重点实验室 北京 100097)

摘 要 驱动漏洞在 Android 手机的安全研究中非常重要,因为驱动运行在内核空间,不仅影响用户的使用满意度,还关系到系统的稳定与安全,但驱动的漏洞挖掘一直都相对较困难,传统模糊测试技术对目标程序缺乏理解、测试随机且盲目的缺点无法适应 Android 驱动漏洞挖掘的需求.通过改进现有模糊测试技术,提出了基于黑盒测试的遗传算法,利用测试的执行结果指导遗传算法,由遗传算法决定测试用例的参数需要遗传还是变异.从而将有效参数遗传到下一代测试用例,无效参数根据执行结果采用不同的策略进行变异,使模糊测试用例可以较快地收敛到有效的范围.为加快漏洞挖掘速度,引入并扩展了参数优化技术,将由遗传算法得到的有效参数进一步修改为特殊数据或使用者预设的数据,更快地达到测试目的.最后基于该算法设计并实现了 Android 驱动的模糊测试系统 Add-fuzz(Android device driver fuzz),利用该系统在多个不同版本的 Android 手机进行了系统测试,挖掘出了 9 个 Android 设备驱动程序的未知安全漏洞.与其它相关测试方法对比,实验结果表明该算法的有效性和适用性表现更优.

关键词 遗传算法;模糊测试;设备驱动;Android;黑盒测试;漏洞挖掘

中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2017.01031

Android Driver Vulnerability Discovery Based on Black-Box Genetic Algorithm

HE Yuan[1),2)] ZHANG Yu-Qing[1)] ZHANG Guang-Hua[3)]

[1)](*Department of National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing* 101408)

[2)](*Department of College of Mathematics and Computer Science, University of Dali, Dali, Yunnan* 671003)

[3)](*Beijing Key Laboratory of IOT Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100097)

Abstract The vulnerability of device drive is an especially important issue for security of Android phones, because device drive run in kernel. Not only it will affect the user's satisfaction, but also concerns the stability and security of the system. Compare with other vulnerability discover the work of device drive is a difficult task. However, there is a lack of understanding about the target program, and the testing is random as well as blind in traditional fuzz testing. So this technique cannot meet the requirements on the vulnerability discovery of Android drivers. By improving the existing fuzz testing techniques, a new genetic algorithm based on black-box test is presented in this paper. The genetic algorithm is performed according to the execution results, and is used to determine whether the parameters of test cases should be preserved or transformed. In this way, valid parameters are passed to the next generation of test cases, while invalid parameters are transformed by different strategies according to the execution results. Therefore, the fuzz test cases can quickly converge to an effective scope. In order to raise the speed of vulnerability discovery, a parameter optimization technology is introduced and expanded. For faster testing,

收稿日期:2016-04-27;在线出版日期:2016-11-09.本课题得到国家自然科学基金(61572460,61272481)、物联网信息安全技术北京市重点实验室开放课题资助.**何 远**,男,1977 年生,博士,实验师,主要研究方向为信息安全、漏洞挖掘.E-mail:hey@nipc.org.cn.**张玉清**,男,1966 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为网络与信息系统安全、漏洞挖掘等.**张光华**,男,1979 年生,博士,副教授,主要研究方向为信任管理、无线网络安全.

the invalid parameters which are obtained from the genetic algorithm are further modified to some special data or user's default data. At last, based on this algorithm, we design and implement a fuzz testing system for Android drivers, which is denoted as Add-fuzz (Android device driver fuzz). We deployed the Add-fuzz on many different versions of Android phones to perform a system testing, and 9 unknown security vulnerabilities about Android device drivers was discovered. Compared with other related works, the experiment results demonstrate that this algorithm has good effectiveness and applicability.

Keywords genetic algorithm; fuzz test; device driver; Android; black-box test; vulnerability discovery

1 引 言

Android 智能手机在生活中得到广泛应用,IDC 预计 2019 年 Android 市场份额将会突破现有 81% 达到 82.6%,全球 Android 手机数量将超过 15.4 亿部①. 与此同时 Android 漏洞数目也不断攀升,截止到目前 National Vulnerability Database(NVD)披露的 Android 漏洞已达 2000 多条②,但已披露的漏洞中与 Android 驱动相关的漏洞只有 26 条,这一数据表明目前 Android 驱动漏洞挖掘研究还没有受到应有的重视.

与传统设备驱动程序一样,Android 设备驱动程序运行在内核空间,其挖掘难度相对较大. 同时 Android 设备驱动漏洞的危害程度、表现形式与传统 PC 驱动漏洞有很大不同. 例如 CVE-2013-6123 漏洞③,由于驱动程序中函数存在数组索引错误,导致攻击者可通过摄像头设备节点获取特权,造成权限提升,使原本看似与安全无关的设备产生了严重的安全隐患. 手机中往往存有大量个人信息和金融信息,一旦驱动中的未公开漏洞被攻击者利用,将产生巨大的危害,所以 Android 驱动漏洞挖掘对保障系统安全具有十分重要的意义.

Android 内核继承于 Linux 操作系统,并对 Linux 内核进行了修改和扩展. 为适应移动终端的需求新增了 IPC binder、low memory killer、ashmem、电源管理等多个 Android 专有驱动. 与此同时各个手机厂商为了体现自己与其它厂商的不同,对手机进行了定制. 例如三星、华为、小米等都对 Android 原生系统进行了不同程度的改动. 虽然 Android 属于开源系统,但不同厂家开源情况参差不齐,驱动源码往往无法及时有效获取,使得 Android 驱动漏洞的挖掘变得尤为复杂.

目前的相关研究主要集中在对 Linux 或 Windows 驱动漏洞的挖掘. 驱动程序运行于系统内核态,静态分析很难分析设备驱动和内核的交互;动态分析需要运行相关硬件并提供非常规的输入,导致驱动漏洞的挖掘难度加大. 文献[1-3]利用符号执行方法对驱动漏洞挖掘进行研究,使用硬件虚拟化技术,解决了需要具体设备测试驱动漏洞的问题. 但符号执行存在路径爆炸问题,挖掘类型单一,更严重的是,目前还没有成熟的 Android 全符号化系统,该技术也不能有效移植到 Android 驱动漏洞挖掘. 在漏洞挖掘方面,模糊测试是快捷、简便的挖掘工具之一,但传统的模糊测试存在对目标程序缺乏理解、测试随机且盲目的缺点,如何克服这些缺点一直是模糊测试研究的重点. 基于遗传算法的模糊测试技术,通过编码将输入数据映射到基因空间,利用白盒测试的方法获取路径条件,以路径覆盖为基础来计算测试用例的适应度,通过遗传算法得到满足条件的测试用例. 没有源码就无法知道程序的执行路径条件,所以在无法获取 Android 驱动程序的源码时,该方法显得力不从心. 如何让漏洞挖掘系统在没有源码的情况下,可以生成更多的有效测试用例,可以在更少的时间内挖掘出更多的未知漏洞是 Android 驱动漏洞挖掘技术的研究难点.

本文系统地研究了针对 Android 驱动漏洞挖掘的模糊测试技术,其主要贡献如下:

(1)实现了基于黑盒测试的遗传算法,无需掌握源码就能完成对测试目标的理解,在测试用例生

① IDC:预计 2019 年 Android 市场份额将达 82.6%. http://www.199it.com/archives/414192.html?url_type=39. 2016,3,18

② https://web.nvd.nist.gov/view/vuln/search-results?query=driver+Android&search_type=all&cves=on. 2016,3,18

③ CVE-2013-6123. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-6123. 2016,3,18

成方面,利用当前个体的测试结果指导遗传算法,从而快速生成有效的测试用例,使模糊测试的效率有较大提高.

(2)漏洞挖掘方面,引入并扩展了参数优化技术,使测试用例在满足有效性的基础上,可按使用者的意图优化相关参数,使漏洞挖掘更具针对性,提升了模糊测试的漏洞挖掘能力.

(3)实现了 Android 驱动漏洞的模糊测试系统 Add-fuzz. 首次较为完整地测试了多个不同 Android 版本的驱动程序,挖掘出了 9 个 Android 驱动的未知安全漏洞.

本文第 2 节对 Android 驱动和模糊测试的相关工作进行论述;第 3 节介绍模糊测试在 Android 驱动漏洞挖掘的可行性及不足;第 4 节提出基于遗传算法的 Android 驱动模糊测试系统 Add-fuzz;第 5 节对 Add-fuzz 进行实验测试并对结果进行分析;第 6 节对 Add-fuzz 系统相关问题进行讨论;第 7 节进行总结.

2　相关工作

模糊测试技术是 1989 年威斯康星州的麦迪逊大学 Miller 教授[4]为了测试 Linux 系统的健壮性而发明的. 多年来在漏洞挖掘方面,针对各种挖掘对象研究者提出了不同的模糊测试系统,例如文献[5]提出了对文件的挖掘系统,文献[6]提出了一种针对网络协议的挖掘系统,文献[7]提出了一种模糊测试通用框架等. 文献[8]则解决了模糊测试过程中校验和导致无法正常挖掘的问题. 文献[9]对模糊测试所面临的种子选择与优化等问题从理论上进行系统的研究. 文献[10]结合静态分析、符号执行等获取执行路径,提高了代码覆盖率,使模糊测试演变成为白盒测试技术. 以 peach① 为代表的"智能"模糊测试,在测试用例生成时可以基于生成的方式也可以基于突变的方式,使其可以适应多个测试场景. 但这些系统所针对的挖掘对象与驱动程序有所不同,无法直接利用已有的模糊测试工具挖掘驱动程序的漏洞.

Io Control Fuzzer② 在 Windows 系统上实现了驱动程序的模糊测试,采用了中间人式的模糊测试技术. 在应用程序调用 Windows 驱动程序时,利用 hook 技术截获相关函数,识别出 Device Io Control 的请求对象是待测驱动时,按照模糊测试策略对该函数调用的参数进行篡改,将篡改后的参数传递给原始的 Device Io Control 函数,然后监控内核是否产生异常,从而挖掘出对应漏洞. 对 Android 系统的 hook 技术有 X-posed③ 等,但对驱动调用的 hook 较为复杂,同时驱动调用需要应用程序进行触发,虽然有 monkey runner④ 等用户界面的自动化测试工具,但应用程序每次测试触发的驱动模块数目有限,不同的界面触发的驱动可能都一样,使得驱动漏洞的挖掘显得效率低下.

在 Android 漏洞的模糊测试方面,最早在 2009 年由柏林工业大学的 Mulliner [11]对短消息的安全漏洞进行测试;到了 2012 年 Mulliner[12] 使用插桩的方法对 Android 系统的 NFC 模块进行模糊测试,并取得了一定的成果. 文献[13-14]通过构造含有畸形数据的 intent,对 Android 组件间通信进行模糊测试,提出了不同的模糊测试系统. 文献[15]对新的虚拟机(art)的相关安全性能进行了模糊测试. 文献[16]针对 Android 系统和 Windows Phone 系统等移动终端平台,对 NFC 第三方应用程序进行了系统和全面的测试,采用模拟标签,并通过进程操作模拟"触碰"操作,实现了自动化的漏洞挖掘,并发现了"触碰"后造成蓝牙、WiFi 自动打开等未知漏洞. 但这些技术无法有效用于 Android 驱动的测试.

Zhou 等人[17]最先对 Android 定制内容的安全问题进行了系统研究,发明了 ADDICTED 系统,通过动态分析设备敏感操作的相关文件,并与原生系统(Android Open Source Project)的相关文件对比,查看其安全保护与原生系统之间的差别. 在多个平台测试后发现了不同厂商的定制系统存在不同程度的安全问题. 虽然测试出了一些定制相关的驱动漏洞,但没有对 Android 驱动漏洞挖掘进行具体研究. 测试方法过分依赖于 Android 原生系统,如果原生系统自身存在漏洞,则该系统无法测出相关漏洞.

I know this⑤ 是 Google 的一个基于系统调用的漏洞挖掘工具,但截止 2012 年 Google 就停止了该工具的更新. Trinity⑥ 是 Jones 发明的漏洞挖掘系统,通过系统调用来挖掘 Linux 漏洞,其主要思想是尽量减少无用的测试,通过随机发送文件描述符

① Eddington M. Peach Fuzzing Platform [Online]. Available: http://peachFuzzer.com. 2016,4,18
② iOCTL Fuzzer v1.2. Fuzzing Tool for Windows Kernel Drivers http://www.darknet.org.uk/2010/12/ioctl-Fuzzer-v1-2-Fuzzing-tool-for-windows-kernel-drivers/. 2016,3,18
③ xposed. http://www.repo.xposed.info/. 2016,3,18
④ monkeyrunner. http://developer.Android.com/reference/Android/view/KeyEvent.html. 2016,3,18
⑤ Ormandy T. Iknowthis. https://code.google.com/p/iknowthis/. 2016,4,18
⑥ Jones D. Trinity: A Linux kernel Fuzz tester. Linux Conf Australia, 2013. http://codemonkey.org.uk/projects/talks/LCA2013-Trinity.pdf. 2016,4,18

和相关参数给系统函数,并根据系统函数的规范对参数进行优化,来降低模糊测试的盲目性.通过交叉编译后 Trinity 可以在 Android 手机进行驱动漏洞挖掘.但 Trinity 不是针对驱动漏洞挖掘来开发,对驱动程序相关函数只优化了部分结构体,没有考虑执行结果对模糊测试的指导作用,Jones 本人也明确提出该工具还有待进一步完善.文献[18]利用组合测试技术对 Trinity 进行了改进,并在理论上证明了其可行性,但每个参数的有效范围在没有源码时无法有效获取,作者也没有实现具体的漏洞挖掘原型系统.

遗传算法[19]最早是由 Bagley 博士提出的,是一种随机搜索与优化算法.文献[20-22]引入了遗传算法用于模糊测试用例的生成.算法采用适应度函数对个体进行评价,由适应度进行优胜劣汰,方法简单,便于设计.但所实现的测试都基于有源码的情况,而且只能对相对简单的某几类程序具有实用性,不能用于驱动程序这类复杂的代码.

多维模糊测试技术对多个输入参数同时变异,它可能会挖掘到单维模糊测试技术无法挖掘到的漏洞,但是多维模糊测试技术会带来组合爆炸问题.文献[23-25]对多维模糊测试技术进行了不同程度的研究,通过对脆弱语句进行有方向的多维模糊测试而触发目标程序中潜在的漏洞.文献[26]对多种多维 Fuzzing 技术进行了研究和比较,总结出多维 Fuzzing 技术的 3 个基本步骤,最后给出了多维 Fuzzing 技术的进一步发展方向.对 Android 驱动漏洞的挖掘某种程度上也属于多维 Fuzzing 技术,但由于驱动的参数相互关联性较大,没有较好的方向性,无法很好的用现有多维 Fuzzing 技术进行漏洞挖掘.

与以上方法不同的是本文对模糊测试结果自动进行分析,根据执行结果采取不同的遗传变异策略,把有效参数遗传到下一代测试用例中,对无效参数不是简单的随机变异,而是根据执行结果有针对性的变异.在遗传变异的过程中无需了解源码,既保留了模糊测试的黑盒测试属性,又能使模糊测试过程较快地收敛于有效测试用例范围,解决了模糊测试盲目生成测试用例的问题.

3 驱动模糊测试技术及分析

模糊测试的主要思想是给系统(程序)发送包含非预期参数,使系统(程序)发生异常,从而发现系统的漏洞.Android 系统多运行于 arm 架构,对其进行漏洞挖掘需要考虑与传统 x86 等平台的差异,幸运的是通过交叉编译即可解决差异问题,其困难在于如何提升驱动程序的模糊测试用例的有效性.

3.1 Android 驱动程序模糊测试

对 Android 驱动程序而言,操作系统提供了相关系统函数供应用程序调用,这些系统函数可以作为模糊测试的入口函数.驱动程序相关的系统函数主要有 open、close、read、write、ioctl 等,完成硬件设备的打开、关闭、读、写操作;ioctl 是设备驱动程序中对设备的 I/O(输入输出)通道进行管理的函数,完成对设备的控制,实现对应功能.例如对相机像素的修改、曝光度的调节等等.对驱动程序分析后发现其主要控制功能往往通过 ioctl 函数实现,ioctl 函数在驱动程序代码中所占比重(代码覆盖率)相对较高,与其它函数交互较多,出现漏洞的可能性较大,所以选取 ioctl 作为模糊测试的入口函数.

不同 Linux 版本的 ioctl 函数原型有一定差异,Linux 2.6.36 后 ioctl 的原型为 int ioctl(int fd, unsigned int cmd, unsigned long arg).其中 fd 是使用 open 函数打开驱动设备文件返回的文件描述符,cmd 为设备的控制命令,是一个 32 位的整数,内核通过特定算法从 32 位整数中提取出对应驱动参数和命令,参数用于鉴别身份,命令实现控制功能.arg 是一些补充参数,控制命令需要参数时通过 arg 进行传递.

传统模糊测试需要随机构造出 ioctl 函数的参数,利用构造出的参数调用 ioctl 函数,然后对系统进行监控,直到系统发生异常,就可以实现对 Android 驱动的漏洞挖掘.其流程如图 1 所示.

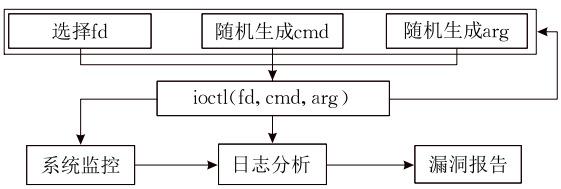

图 1 传统模糊测试驱动漏洞挖掘流程

针对某个驱动的 ioctl,例如对手机前摄像头驱动进行测试,其 fd 为打开/dev/video0(不同系统命名有所不同)的文件描述符.随机生成 cmd 和 arg 参数.用生成的参数构造 ioctl 函数调用,即可进行模糊测试.其测试结果如图 2 所示.

```
[44438] ioctl(fd=257, cmd=0xba374507, arg=0x613a)=-1(ENOTTY)
[44439] ioctl(fd=257, cmd=0xeeec4507, arg=0x74)=-1(EINVAL)
[54371] ioctl(fd=257, cmd=0x8004450, arg=0xc0008000)=0
```

图 2 传统模糊测试结果样本(部分截取)

图 2 中[44438]是测试用例的序号;fd=257 是文件描述符对应的编号;cmd 和 arg 为测试用例的参数;执行结果等于-1 代表参数出错,并给出错误类型,等于 0 代表测试用例被正确执行.

3.2　传统模糊测试对驱动漏洞挖掘的不足

在以下分析中为论述方便定义两个概念:

(1)有效(参数). 参数满足函数特定的规范或属性即为有效. 如参数范围为不大于 32 位的二进制整数,则参数在$(0,2^{32})$都为有效.

(2)无效(参数). 与有效参数对应,参数不满足函数特定的规范或属性. 如参数范围为不大于 32 位的二进制整数,则参数大于2^{32}即为无效.

使用 ioctl 函数进行模糊测试,其参数需要满足以下条件:

(1)fd 有效,即 fd 必须是文件描述符而不能为其它. 如果 fd 是无效字段,或者无权限打开对应文件,则函数调用会被拒绝.

(2)cmd 有效,即 cmd 必须为$(0,2^{32})$之间的二进制正整数. 为避免误操作,cmd 会实现身份鉴别功能,通过算法从 cmd 提取出的身份标识与 fd 一一对应;cmd 解析后所执行的操作是对应硬件驱动的操作. 不满足参数有效性要求,函数调用将停止,并抛出异常结果.

(3)当驱动需要参数完成控制操作时,参数由 arg 提供,一般对应整型、指针、结构体等类型. arg 需要满足类型一致和其它要求的属性.

遗憾的是,参数的有效性在运行阶段才能进行检测,在没有驱动相关源码时,无法提前得到对应的参数规范,所以无法在编译前对参数有效性进行优化,无效参数在测试时,无法通过有效性检查,测试将被迫停止,因此无法执行相关的驱动程序语句. 传统模糊测试往往只对单个参数进行变异,参数空间较小,在一定范围内可以有效挖掘漏洞. ioctl 函数调用涉及 3 个变量同时变异,在驱动程序的测试过程中,fd 可以精确获取到;cmd 为 32 位整数,对应2^{32}种组合. 满足 fd 和 cmd 有效性后,需要 arg 不断变异,同样以 32 位整数为例,需要遍历完2^{32}种有效参数,因此一个驱动漏洞的检测需要测试完2^{64}种不同组合,组合爆炸导致传统模糊测试在有限时间内几乎不可能测试出驱动漏洞.

4　基于遗传算法的 Add-fuzz 系统设计与实现

模糊测试的目的是测试模块内部的处理逻辑是否有漏洞,通过对传统模糊测试的参数分析可知,使用无效参数的结果是还没进入到模块内部就被拦截(或中断)在函数外围了. 因此需要对传统模糊测试进行改进,以适应 Android 驱动程序的模糊测试.

在测试过程中,通过分析传统模糊测试的结果发现,随机生成的测试用例很多都是在重复执行同一种错误类型的数据,或者是重复执行相同的条件路径. 虽然执行 ioctl 后有结果提示,但是传统模糊测试不能利用执行结果指导后续测试用例生成. 遗传算法在数据优化方面具有较好的性能,所以利用遗传算法对传统模糊测试进行改进. 模糊测试需要测试样本具有多样性,但在测试驱动程序内部的处理逻辑是否有漏洞时,部分参数的多样性阻碍了测试的顺利进行. 遗传算法是一种优化算法,可以使参数收敛到局部最优或全局最优,正好可以弥补参数无效的缺点.

为了满足改变传统的模糊测试的需求,所面临的挑战主要有:

(1)如何利用历史执行的提示信息指导后续的参数生成?

(2)如何在减少测试用例的基础上,更快地挖掘到漏洞?

对问题 1,充分利用测试结果,结合遗传算法,并对遗传算法进行改进,使其更简洁,同时还能满足对测试用例进行遗传变异的需求. 详细设计见 4.2 节.

对问题 2,利用现有漏洞得出一些漏洞模式,通过对这些漏洞模式的理解,将漏洞模式转化为对应的参数,在保证参数有效性的前提下,尽可能多地利用设置的特定参数. 详细设计见 4.3 节.

4.1　改进后的模糊测试系统总体设计

通过对传统模糊测试结果样本的分析,可知执行结果有提示信息,进一步查看 ioctl 函数的具体实现,其形式如下:

```
static int ioctl(struct inode *inode, struct file *filep,
unsigned int cmd, unsigned long arg)
 {…
if (_IOC_TYPE(cmd) !=XX_IOC_MAGIC)
return EINVAL;
switch(cmd)
{case xxx:
 {…
 if (copy_from_user(&nc, (const char*) arg,
     sizeof(nc)) !=0)
 return -EFAULT;
 …
}
```

不同 ioctl 的实现有较大差异，但梳理后发现其形式基本相同，首先对 cmd 进行有效性判断，然后执行对应的代码，在执行代码时对 arg 参数进行判断. 所以从执行结果的提示信息可以看出参数的有效性.

设计时一方面通过利用测试结果配合遗传算法，缩小参数的取值范围，从而生成新的满足有效性要求的测试用例，使测试用例尽快收敛到有效参数空间；另一方面，在生成有效测试用例的基础上，对参数可变部分按已知漏洞特征等进行优化，这样可以有效提高模糊测试的命中率. 经过改进后的 Add-fuzz 的工作流程如图 3 所示.

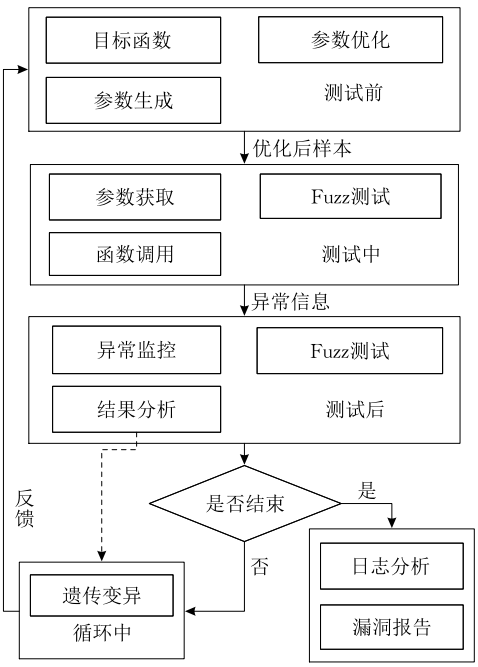

图 3 Add-fuzz 流程图

测试前，选取目标函数 ioctl，随机生成测试数据，分别完成 fd 选择、随机生成 cmd 和 arg 参数. 测试进行后参数的生成配合遗传变异策略完成. 参数生成后按已知漏洞类型或相关规则对生成的参数进行优化.

测试中，将经优化后的数据样本发送到模糊测试模块，模糊测试模块接收到参数后调用对应的函数进行测试.

测试后，记录模糊测试的测试用例参数，对测试结果进行记录分析，同时对系统进行监控，完成对异常情况的记录. 对系统的异常监控可以利用系统日志.

如果模糊测试达到预定的目标，如手机重启、死机等，或者测试到达一定条件，如达到测试数目的上限或限定的时间，则程序停止. 对相关日志进行分析，然后输出漏洞报告. 否则继续执行.

继续执行时，根据测试结果分析指导遗传算法生成下一代的测试用例，不断循环，直到程序停止.

4.2 基于结果反馈的黑盒遗传算法

当一个模糊测试用例被执行后，若执行结果为 0，表明 fd 和 cmd 和 arg 参数都有效，下一步只需对 arg 参数值进行变异即可. 当系统调用执行结果为 −1 时，表明参数未通过函数检验. ioctl 函数执行结果主要有 EBADF、EFAULT、EINVAL、ENOTTY 几种. EBADF 表示 fd 必须为打开的文件描述符. 确定相关的驱动后，打开对应的文件描述符传递给 ioctl 函数，直接使用有效参数进行测试. EFAULT 代表 arg 指向错误的内存地址，应对 arg 的内存地址进行变异. EINVAL 代表 cmd 或 arg 参数错误，需对 cmd 或 arg 进行变异. ENOTTY 代表 cmd 与 fd 的对应关系不符，需要对 cmd 进行变异.

改进后的算法首先利用 ioctl 函数的执行结果生成一个新的遗传算法. 其遗传变异机制如式(1)所示.

$$\begin{cases} x_{i+1}=cmd_i+arg_{i+1}, & result=0 \\ x_{i+1}=g(cmd_i)+f(arg_i), & result=-1 \end{cases} \tag{1}$$

其中 f 代表按一定的概率对 arg 进行变异的函数；g 代表按一定的概率对 cmd 进行变异的函数；x_i 代表当前个体；x_{i+1} 代表下一代个体.

然后利用 ioctl 函数的执行结果和错误提示信息指导下一代测试用例参数的遗传和变异.

具体运算过程如下：

(1) 初始化. 随机生成 1 个初始个体或者使用特定的初始值，然后执行函数调用，得到执行结果，第二次随机生成新的个体. 这 2 个个体组成初始群体 $P(0)$，开始执行遗传算法. 也可以增加群体的数目，但为了计算方便，直接压缩到 2 个个体.

(2) 个体评价. 计算个体的适应度，利用测试结果来评价个体. 测试结果的提示信息可以粗略对应代码覆盖率. 无论执行结果是否正确，模糊测试要求下一个测试用例有所改变，所以随机生成的下一代个体默认具有较高适应度.

(3) 选择运算. 初始时，选择初始个体和下一代个体. 从第二代个体后可以随机选择当前个体和下一代个体.

(4) 交叉运算. 交叉算子作用群体，交叉位置固定在 cmd 和 arg 中间. 模糊测试每次都需要新的个体，所以交叉率直接选取为 1.

(5)变异运算. 根据执行结果采取不同的变异策略,随机生成的下一代个体虽然默认具有较高适应度,但是因为参数随机生成,仍可能无法满足有效性要求,根据当前个体的执行结果指导新个体进行变异. 从而得到最优个体.

(6)群体 $P(i)$经过选择、交叉、变异运算之后得到下一代群体 $P(i+1)$. 在下一代群体中只遗传(保留)了有效参数,并对无效参数进行变异,这个新的下一代个体,即为前后 2 代个体遗传变异后的最优解,将作为下次测试的当前个体进行模糊测试. 其它个体不是最优个体将被丢弃.

对算法改进后,有效参数得到遗传,无效参数得到变异,遗传变异时根据结果的错误提示信息指导对应的操作. 其对应算法如算法 1 所示.

算法 1.　改进后的遗传算法.

输入:第 i 代 ioctl 参数及执行结果和第 $i+1$ 代 ioctl 参数

输出:新的第 $i+1$ 代 ioctl 参数

```
Agfunction(fd,cmd[i+1],cmd[i],arg[i+1],
           arg[i],result, result_type)
IF result=0
  cmd[i+1]=cmd[i]        //遗传 cmd
  arg[i+1]=arg[i+1]      //随机新 arg
ELSE SWITCH (result_type)
CASE EFAULT
  cmd[i+1]=cmd[i]        //遗传 cmd
  arg[i+1]=f(arg[i])     //按 f 函数变异 arg
BREAK;
CASE ENOTTY
  cmd[i+1]=g(cmd[i])     //按 g 函数变异 cmd
  arg[i+1]=arg[i]        //遗传 arg
BREAK;
CASE EINVAL
  cmd[i+1]=cmd[i+1]      //随机新 cmd
  arg[i+1]=f(arg[i])     //按 f 函数变异 arg
DEFAULT:
    …BREAK;
```

模糊测试的代码覆盖率越高,不代表发现漏洞越多,但代码覆盖率越高发现漏洞的可能性会越大. 所以代码覆盖率是计算测试用例适应度的理想指标. 代码覆盖率通常需要通过源码才能计算. 对 ioctl 等系统函数来说,代码覆盖率恰好能与执行结果的提示信息粗略对应,如图 4 所示,因此直接利用执行结果就可以达到评价个体适应度的目的,省去了计算过程,使得遗传算法变得更为精简,不需要具体的程序源码就能评价测试用例的适应度.

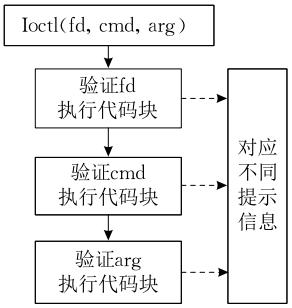

图 4　执行结果提示与代码覆盖率关系图

以 3.1 节图 2 中的测试用例样本[44438]为例,结果为−1 代表函数调用没有正确执行,ENOTTY 代表 cmd 参数错误,所以程序执行到验证 cmd 就终止了. 为了便于计算,假设每个参数验证过程对应一个基本块,并假设驱动程序总共有 10 个代码块. 则上述测试用例覆盖了 1 个条件路径的代码块,代表 fd 通过验证,代码覆盖率为 0.1;同理,图 2 中样本测试用例[54371]中,执行结果为 0 代表函数调用成功执行,测试用例覆盖了 3 个条件路径的代码块,代表 fd、cmd、arg 都通过验证,代码覆盖率为 0.3. 因此可以直接由执行结果来评价测试用例. 为了程序的简化,直接由执行结果对应的提示信息计算适应度.

当执行结果为 0 时,表明参数达到局部最优,代码覆盖率最高,也就是说参数有效覆盖了对应的条件路径,但测试后没发现异常或漏洞,所以在保证 arg 有效的前提下对 arg 的值进行变异. 依据文献[27]的研究结论,进化次数取值参考区间为 100～1000 较为合理,在此选取 500 作为参考值,可根据不同测试粒度选取不同参考值,其作用是跳出局部最优解. 当参数收敛到最优解时,变异 arg 的值测试 500 次后仍没发现异常或漏洞,则退出目前的有效参数,对 cmd 进行变异,继续测试 ioctl 对应的其它路径,从而跳出局部最优解. 对驱动程序的分析可知,一个驱动程序对应的 cmd 计算出的控制参数值在一个连续的范围,当 cmd 已达到最优时,满足参数有效性要求,所以对 cmd 的变异只需要在目前参数值的基础上以步长为 1 进行变异. 随后根据结果采取不同变异策略,再次进入到遗传变异的测试过程. 整个遗传变异过程 2 次使用当前个体的执行结果指导遗传算法,确保遗传变异后的个体具有最高的概率保持参数最优,加快了参数收敛到有效范围的速度.

上述算法中下一代个体是随机生成的,因为随机生成的数据很大程度上仍然不能满足有效性要求,根据当前执行结果,对新个体进行变异. 函数 f、g 是在生成下一代个体的基础上根据执行结果提示

对参数进行变异,从而降低下一代个体出现上一代错误的概率.函数 g 完成对 cmd 字段按参数的生成规则进行变异,改变 cmd 的"魔数"字段的值,或改变控制字段的值,使其更快满足 cmd 参数有效性要求.函数 f 完成对 arg 参数的变异.当执行结果表明 arg 参数无效时,将下一代个体的 arg 字段对应的地址变异为非空地址等,使 arg 参数以更快的速度达到有效性要求.遗传变异过程如图 5 所示.

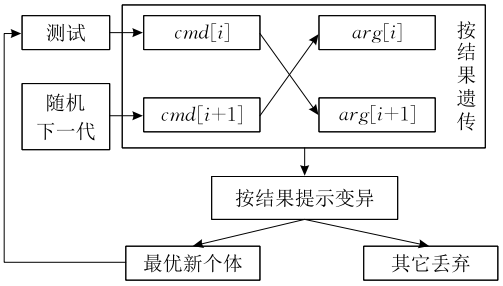

图 5 参数遗传变异流程图

改进后的遗传算法与传统遗传算法区别在于:

(1)初始种群压缩到 2 个个体.传统遗传算法需要有一定数量的初始数据,通过对每个个体计算适应度,然后判断是否满足输出,不满足则继续进行选择、变异、交叉遗传操作,产生新的种群,直到满足停止条件.从初始群体不断演变,一直到生成最优个体.改进后的遗传算法确保每次只有 2 个测试用例参与运算,即当前个体 x_i 和新的个体 x_{i+1},并生成最优的新的下一代个体,生成的新后代包含 2 个新个体(加上原有母体则有 4 个个体),这 2 个后代的 cmd 有一个来自执行过的上一代个体,其代码覆盖率与之相同,如果代码覆盖率高于 0.2,即参数通过了 cmd 验证,则选择该个体作为最优个体,否则,选择含有新 cmd 的个体作为最优个体.如此循环,最后使得参数收敛到有效范围.

传统遗传算法是按总体来考虑,通过对初始种群的选择、交叉、变异得到最优个体.改进后的遗传算法是以部分来考虑,每次得到新个体后执行对应的测试.并不断利用测试结果指导新个体的生成,所以每次都能结合当前执行结果得到当前最优解,使问题由全局最优解分解为局部最优解,把原有全集中的漏洞挖掘,变为集合中每一部分的可能漏洞的挖掘.

传统遗传算法对初始值较为敏感,改进后的算法初始值对进化的影响较小,如果初始值直接为有效参数,则测试将在 500 次后变异为其它参数,继续对驱动程序进行挖掘.反之,初始值为无效参数,则按算法进行遗传变异,直到参数收敛为有效参数,然后重复有效参数的测试过程.换句话说,在极端情况下所影响的最大值测试次数就是 500 次(进化的次数的选取值).

(2)利用执行结果代替适应度函数实现个体评价.现有遗传算法需要源码支持,或者需要插桩等技术辅助,得到条件路径后选取适应度函数,然后代入参数计算个体适应度.在测试数据的生成过程中,遗传算法经常会因为适应度的不同设计而出现收敛速度慢、过早收敛、局部收敛等问题.改进后的算法保持了模糊测试的黑盒测试特性,利用执行结果评价个体,避免了因收敛而造成的问题.

(3)算法简单有效,使执行速度加快.变异函数等与传统遗传算法形式上有所不同,但算法利用了遗传变异和优胜劣汰的思想,能有效解决过多测试用例参数无效的问题.传统遗传算法中的交叉变异是为了改善群体的多样性,避免陷入次优解.此处使用随机参数达到多样性的目的.利用变异函数使参数满足特有属性的要求.

通过遗传算法,使参数可以较快地收敛到有效范围,从而使测试深入到函数内部,达到减少无效测试用例的目的.

4.3 测试用例参数优化

遗传变异机制针对参数的有效性进行优化,完成了模糊测试能打入函数内部的任务.利用遗传算法不仅使 cmd 更快达到有效值,而且使 arg 参数的属性得到满足,例如 arg 参数的地址不能为空,arg 内存满足 64 位对齐等特有属性.但漏洞往往由异常参数造成,在满足参数有效性的基础上,对参数的值没有特定的规范,需要测试"意外"数据造成的漏洞.

由已披露的漏洞,明确知道一些特殊值可能会导致已知漏洞的发生,模糊测试时希望这些值能被测试到,则在生成测试用例后,随机将参数替换为特殊值,这样就能发现意料中的漏洞.例如为了检测缓冲区溢出漏洞,填充超长字符串即可.为了便于日志分析,重复填充相同字符串,如"ABAB"、"AAAAAAAAAAAAAAAAAA"等.对于字符处理漏洞,将参数优化为" ","null"、"*"、"#"等非字母字符.对于整数溢出漏洞,填充整数的边界值,如 −1、0xFFFFFFFF 等.

对特殊情况,可以根据需求,增加对应的优化规则.例如 Android 内核发生空指针解引用等漏洞时,手机会重新启动.继续挖掘其它漏洞需要重新执行

模糊测试程序,由于执行参数相同,程序仍会再次触发对应的漏洞,导致挖掘无法深入,这时对相关参数进行优化,当测试用例的 cmd 与已测出漏洞的 cmd 相同时,修改 cmd 为其它值. 如果还想深入测试该 cmd 对应的其它漏洞,则生成测试用例参数后,强制使 arg 不等于已测出漏洞的 arg,从而可以挖掘出同一模块的不同漏洞,实现了一定程度的深度挖掘.

遗传算法使参数有效性得到提高,参数优化可以挖掘出已知类型的漏洞. 但模糊测试的目的不是测试所有的有效参数,也不仅仅是挖掘出已知类型的漏洞,还需要挖掘未知的漏洞. 因此在实际使用过程中,可根据具体情况选取不同的随机函数执行遗传算法.

通过上述遗传算法和优化机制,整个模糊测试过程完成了一个学习反馈的过程,并利用已知漏洞类型进行数据优化,测试用例参数由随机生成变为已知有效数据、故意错误数据和随机数据的组合,大大减少了测试的盲目性.

该遗传算法主要以 ioctl 函数进行论述,但算法的适应性不局限于 ioctl 函数,只要与 ioctl 函数有相似的性质就可以使用所提出的算法.

5　实验验证与结果分析

为评估 Add-fuzz 的有效性,通过实验验证并与 Trinity 和 peach 进行了相关对比. 主要从测试用例的有效性、挖掘性能两方面对 Add-fuzz 系统进行验证分析.

5.1　实验环境

实验主要在三星 GT-I9300 Android4.1(内核版本 3.0.31)手机进行对比测试,并在华为 mt1-u06 android4.1(内核版本 3.0.8)、小米 4 Android4.4(内核版本 3.4.0)、三星 GT-N7100 Android5.1(内核版本 3.0.64)、三星 G9280 Android 5.1(内核版本 3.10.6)等进行了相关测试.

5.2　测试用例的有效性

在模糊测试中,代码覆盖率是一个重要指标. 因为执行更多的路径意味着可能发现更多的漏洞. 在代码覆盖率相近的情况下,或者在挖掘出同样漏洞的情况下,测试用例数目越少,代表测试用例有效性越高. 实验中代码覆盖率由测试用例成功执行的数目来替代. 测试用例成功率=成功执行数/测试用例数.

Trinity 与 Add-fuzz 具有部分相似的特点,测试用例的有效性与 Trinity 进行对比,使用与 Trinity 类似的优化策略,并增加遗传算法进行优化,对 viedo1 等驱动程序先使用 Trinity 进行漏洞挖掘,然后用 Trinity 的初值作为 Add-fuzz 的初值进行测试. 在 Trinity 没有挖掘到漏洞而 Add-fuzz 挖到漏洞的情况下,对比没有实际意义,所以只对两者挖掘出相同漏洞的情况下进行对比. 执行结果如表 1 所示.

表 1　测试用例执行情况对比

驱动模块	Add-fuzz		Trinity		Add-fuzz	Trinity
	Css	Cgs	Css	Cgs	成功率	成功率
pn544	36541	2432	250893	239	0.06656	0.00095
video0	24584	1786	184537	172	0.07265	0.00093
video1	25820	1917	203268	198	0.07424	0.00097
bluetooth	35827	2512	150893	143	0.07011	0.00095
ecryptfs	24362	1591	50324	47	0.06531	0.00093
ashmem	9458	683	10039	9	0.07221	0.00090
平均成功率	156592	10921	849954	808	0.06974	0.00095

注:平均成功率=各个漏洞成功率的平均数
Css:代表执行的测试用例数;Cgs:代表执行成功的测试用例数

以 viedo1 驱动程序为例,Trinity 运行 203268 条测试用例后发现漏洞,执行成功的测试用例只有 198 条,成功率约为 0.097%. 对 Trinity 的测试结果分析后,发现 Trinity 有很多无效的测试用例是 fd 和 cmd 随机产生导致的错误,虽然优化了 arg 参数,但测试效率还是很低. 从结果可看出其侧重无效参数对挖掘漏洞的贡献,但没有充分利用测试结果,对测试目标具有盲目性. Add-fuzz 使用了遗传算法后,只用了 25820 条测试用例就成功发现了漏洞,并且执行成功数为 1917 次,成功率接近 7%. 不同驱动程序的测试结果有所不同,但所测试的驱动模块 Add-fuzz 的成功率约为 0.07,同等条件下 Trinity 成功率约为 0.00097. Add-fuzz 测试用例的有效性明显提高.

5.3　挖掘性能分析

在进行性能比较前,先做如下定义.

挖掘能力,指在程序的正常执行条件下(或达到程序的停止条件为止)挖出漏洞的数目.

挖掘效率,指挖出相同的漏洞所需测试用例的数量,测试用例数越少效率越高.

执行速度,指一条测试用例的平均执行时间.

深度挖掘,当挖掘出一个漏洞后是否可以在同样参数下继续挖掘出其它漏洞.

所需知识,指写出高效的优化策略时所需的对测试对象的理解深度. 其它指标则指是否具有对应的功能.

Peach 是目前较为流行的挖掘工具之一,在挖

掘性能方面，分别使用 Peach、Trinity、Add-fuzz 对驱动漏洞进行挖掘. 其结果如表 2 所示.

表 2 驱动漏洞挖掘结果对比

对比项	漏洞数	测试用例	测试时间/s
Add-fuzz	9	234875	32882
Trinity	6	849954	59495
Peach	2	83947	9234

从表 2 可看出 Add-fuzz 和 Trinity 的驱动漏洞挖掘能力要好于 Peach. 在挖掘网络协议漏洞时，Peach 可以基于状态机进行测试，驱动测试没有状态机，Peach 就会退化为随机测试，所以对 Android 驱动 Peach 只挖出了 2 个漏洞(此项指标与测试者水平有一定关系，存在一定的不公平因素)，而 Trinity 和 Add-fuzz 分别挖出 6 个和 9 个未知漏洞. 在效率方面，Trinity 平均一个漏洞约 140000 条测试用例，Peach 约 42000，Add-fuzz 效率最高，平均为 26000. Add-fuzz 执行遗传算法和优化策略，使得计算速度变慢. 三者都用了优化技术，但只有 Add-fuzz 用了基于结果的遗传算法，挖掘漏洞时只需要 Trinity 的约 1/6 测试用例数，虽然每个测试用例的执行时间约为 Trinity 的 2 倍，但总的时间还是比 Trinity 少，挖掘效率更高. Add-fuzz 对 Android 驱动挖掘更深. Add-fuzz 则可以设置优化规则而避免同一漏洞被重复触发. 在使用(开发)方面，Trinity 需要对系统函数有专业了解才能写出好的优化策略，Peach 需要对挖掘对象具有专业知识，Add-fuzz 则不需要驱动的专业知识，只需要对函数执行结果的含义有所了解就能达到较好的效果. 当没有优化策略时，三者都退化为随机测试，都无需专业知识. 具体性能对比如表 3 所示.

表 3 驱动漏洞挖掘性能对比

软件	挖掘能力	挖掘效率	执行速度	遗传变异	参数优化	深度挖掘	所需知识
Peach	弱	中	快	无	弱	无	专业
Trinity	中	低	中	无	强	无	专业
Add-fuzz	强	高	慢	有	强	有	执行结果

小米 4、三星 G9280、GT-N7100 这三款手机对目录文件进行了保护，程序无法正常遍历/dev 目录，导致无法进行模糊测试. 但三星 GT-N7100 的/dev/下的多个文件的权限较低. 例如 dev/video1 只需要 camera 权限，所以通过 GT-I9300 Android4.1 挖掘出漏洞后，在 GT-N7100 上利用 ndk 编程执行相关的漏洞攻击代码，就可以验证对应的驱动漏洞. 通过此方法，挖掘出 GT-N7100 的 3 个驱动漏洞，共 12 个漏洞，但 GT-N7100 的 3 个漏洞与 GT-I9300 属于同一漏洞类型. 对挖掘的漏洞进行分析后发现，漏洞类型主要是空指针解引用，造成手机重启，可以用作拒绝服务攻击；其中还有一个是缓冲区溢出漏洞，可利用漏洞进行提权攻击. GT-I9300 Android4.1 的漏洞如表 4 所示.

表 4 Add-fuzz 挖掘出的 Android 驱动漏洞

手机型号	驱动模块	漏洞类型	造成危害
三星 GT-I9300 Android4.1 内核 3.0.31	pn544	空指针	拒绝服务
	video0	空指针	拒绝服务
	video1	空指针	拒绝服务
	binder	空指针	拒绝服务
	accelerator	空指针	拒绝服务
	bluetooth	空指针	拒绝服务
	ecryptfs	空指针	拒绝服务
	ashmem	空指针	拒绝服务
	video1	缓冲区溢出	权限提升

5.4 算法优点分析

本文提出的系统利用了遗传算法改进了其它系统的缺点，主要优点有以下几个方面：

(1) 充分利用了执行结果对测试用例生成的指导作用，使得在没有源码的情况下，达到快速收敛的目的，在同等条件下挖掘漏洞所需要的测试用例数目明显减少.

(2) 利用测试结果评价个体的适应度，完成有效参数的遗传后，再次利用执行结果的具体错误提示信息指导模糊测试进行参数变异. 计算过程花费了一定时间，但由于测试用例的减少，总体执行速度仍优于其它工具.

(3) 参数优化技术的扩展，可以使用户根据需求增加优化策略，提高了挖掘的灵活性和针对性.

5.5 算法缺点分析

(1) 执行速度慢，Trinity 直接生成数据后就执行，Add-fuzz 因为需要查验结果，并以结果为指导进行变异. 当挖掘到漏洞后，设置需要避免的参数，增加了判断过程，每次执行时间又增长.

(2) 普适性较差，当测试没有执行结果可以参考时，所设计的算法性能将退化. 具体分析参考讨论部分的论述.

6 讨　论

检测出 Android 低版本漏洞后,利用攻击代码在高版本手机进行测试,在部分 Android 5.1 的手机上攻击有效,说明同一驱动会在不同内核版本使用,对系统升级时驱动的漏洞没有引起厂商的足够重视,只是简单求助于 seLinux 的保护功能而不去挖掘并修复相关漏洞.导致公布的驱动漏洞严重偏少,其结果与本文引言部分的调查结果一致.

测试中发现某些漏洞无法重现.例如当某个测试用例导致指针为空,但不解引用,系统一切正常,当后续的测试触发了该空指针的解引用时,将造成系统重启,但此漏洞却无法由崩溃时的日志参数重现.有时会出现测试到一定程度后强行停止,然后使用驱动相关的某个功能就会导致手机重启.也就是说某种漏洞的触发是由当前一个或多个测试用例的积累,再加上后续的某个条件才触发的.因此对这种情况需要对驱动程序详细分析才能确定是否真正存在漏洞.

执行结果 EINVAL 代表 cmd 或 arg 参数错误,通过遗传变异机制能解决 cmd 造成的错误.由于 arg 参数类型不确定,当 arg 为结构体类型等对相关变量类型有要求时,由执行结果无法得到 arg 参数出错的具体字段,会造成算法性能的退化.除了执行结果 EINVAL 对生成测试用例的指导不足外,Add-fuzz 对没有结果提示的程序无法进行高效测试.利用执行结果指导模糊测试是 Add-fuzz 的最大优点,当被测试程序没有执行结果反馈时,改进后的遗传算法无法有效执行,这时只能求助于其它技术(如插桩技术)获取程序路径信息,然后生成对应的适应度函数,否则改进后的模糊测试将会退化到传统模糊测试.与此同时,Add-fuzz 是针对 Android 驱动漏洞挖掘设计的,使用了 ioctl 系统函数调用,对其它系统函数导致的驱动漏洞需要适当修改才能进行测试,需要进一步扩充其它函数才能全面挖掘驱动程序的漏洞.完善遗传算法,并利用机器学习等相关理论,通过对历史数据进行统计学习,从而使算法不依赖执行结果的语义知识,提高算法的普适性,使其在不具有相关语义知识或只有有限语义知识的情况下,获得与基于语义变异测试的同等效果,是下一步计划的研究工作.

Trinity 在挖掘系统调用过程中需要 root 权限,Add-fuzz 无需 root 权限,其适用性更强.但对于具有 seLinux 安全机制的机型,即使获取到 root 权限,也因为无法获得正确的上下文而无法进行正常测试.突破 seLinux 的安全机制进行模糊测试是下一步需要研究的内容.

7 结束语

本文介绍了一种带反馈机制和数据优化的模糊测试技术,通过遗传算法把反馈机制和数据优化相结合.利用执行结果指导测试用例的生成,使得测试过程无效参数的比重减少,提高了模糊测试的效率.应用于 Android 驱动漏洞挖掘实践后,发现了多个未公布的驱动漏洞,如 binder、camera 等可利用的拒绝服务攻击漏洞.与其它测试方法比较的结果表明在 Android 驱动漏洞挖掘方面 Add-fuzz 综合性能更优.但 Add-fuzz 没有突破模糊测试面临的访问控制缺陷;对系统异常的监控粒度较粗,只从 Android 系统的 log 进行监控;其普适性不强等这些缺点需要进一步进行研究和改进,才能使其提升挖掘漏洞的效率.

参　考　文　献

[1] Neugschwandtner M, Milani Comparetti P, Haller I, et al. The BORG: Nanoprobing binaries for buffer overreads//Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. New York, USA, 2015: 87-97

[2] Kuznetsov V, Chipounov V, Candea G. Testing closed-source binary device drivers with DDT//Proceedings of the USENIX Annual Technical Conference. Boston, USA, 2010: 4-5

[3] Renzelmann M J, Kadav A, Swift M M. SymDrive: Testing drivers without devices//Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Hollywood, USA, 2012: 279-292

[4] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Communications of the ACM, 1990, 33(12): 32-44

[5] Kim H C, Choi Y H, Lee D H. Efficient file fuzz testing using automated analysis of binary file format. Journal of Systems Architecture, 2011, 57(3): 259-268

[6] Sun Xin, Yao Yi-Yang, Lu Xin-Dai, et al. Research and implementation of fuzzing testing based on HTTP proxy. Chinese Journal of Netword and Information Security, 2016, 2(2): 75-86

[7] Shudrak M O, Zolotarev V V. Improving fuzzing using software complexity metrics//Proceedings of the International Conference on Information Security and Cryptology. Seoul, Korea, 2015: 246-261

[8] Wang T L, Gu G F. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection//Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 2010: 497-512

[9] Rebert A, Cha S K, Avgerinos T, et al. Optimizing seed selection for fuzzing//Proceedings of the 23rd USENIX Conference on Security Symposium. San Diego, USA, 2014: 861-875

[10] Godefroid P. From blackbox fuzzing to whitebox fuzzing towards verification//Proceedings of the International Symposium on Software Testing and Analysis (ISSTA) 2010. Trento, Italy, 2010: 1-38

[11] Mulliner C, Miller C. Injecting SMS messages into smart phones for security analysis//Proceedings of the 3rd USENIX Conference on Offensive Technologies. Berkeley, USA, 2009: 1-7

[12] Mulliner C, Miller C. Dynamic Binary Instrumentation on Android. Melbourne, Australia: RuxCon, 2012

[13] Sasnauskas R, Regehr J. Intent fuzzer: Crafting intents of death//Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics. San Jose, USA, 2014: 1-5

[14] Yang K, Zhuge J, Wang Y, et al. IntentFuzzer: Detecting capability leaks of android applications//Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. Kyoto, Japan, 2014: 531-536

[15] Kyle S, Leather H, Franke B, et al. Application of domain-aware binary fuzzing to aid android virtual machine testing//Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul, Turkey, 2015: 121-132

[16] Wang Zhi-Qiang, Liu Qi-Xu, Zhang Yu-Qing. Research of discovering vulnerabilities of NFC applications on Android platform. Journal on Communications, 2014, 35(z2): 118-123(in Chinese)
(王志强, 刘奇旭, 张玉清. Android 平台 NFC 应用漏洞挖掘技术研究. 通信学报, 2014, 35(z2): 118-123)

[17] Zhou Xiao-Yong, Lee Yeonjoon, Zhang Nan, et al. The peril of fragmentation: Security hazards in android device driver customizations//Proceedings of the IEEE/SP 2014 Symposium and Workshops on Security and Privacy. California, USA, 2014: 409-423

[18] Garn B, Simos D E. Eris: A tool for combinatorial testing of the Linux system call interface//Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation Workshops. California, USA, 2014: 58-67

[19] Fink G, Bishop M. Property based testing: A new approach to testing for assurance. ACM SIGSOFT Software Engineering Notes, 1997, 22(4): 74-80

[20] Pargas R P, Harrold M J, Peck R R. Test-data generation using genetic algorithms. Software Testing Verification and Reliability, 1999, 9(4): 263-282

[21] Srivastava P R, Kim T. Application of genetic algorithm in software testing. International Journal of Software Engineering and Its Applications, 2009, 3(4): 87-96

[22] Wu Z Y, Atwood J W, Zhu X Y. A new fuzzing technique for software vulnerability mining//Proceedings of the International Conference on Software Engineering. Chennai, India, 2009: 59-66

[23] Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing//Proceedings of the 31st International Conference on Software Engineering. Washington, USA, 2009: 474-484

[24] Berndt D, Fisher J, Johnson L, et al. Breeding software test cases with genetic algorithms//Proceedings of the 36th Hawaii International Conference on System Sciences. Hawaii, USA, 2003: 338-348

[25] Liu G H, Wu G, Zheng T, et al. Vulnerability analysis for x86 executables using genetic algorithm and fuzzing//Proceedings of the 3rd International Conference on Convergence Hybird Information Technology. Busan, Korea, 2008: 491-497

[26] Wu Zhi-Yong, Xia Jian-Jun, Sun Le-Chang, Zhang Min. Survey of multi-dimensional Fuzzing technology. Application Research of Computers, 2010, 27(8): 2810-2813(in Chinese)
(吴志勇, 夏建军, 孙乐昌, 张旻. 多维 Fuzzing 技术综述. 计算机应用研究, 2010, 27(8): 2810-2813)

[27] Zhou Ming, Sun Shu-Dong. Genetic Algorithm Principle and Application. Beijing: National Defence Industry Press, 1999 (in Chinese)
(周明, 孙树栋. 遗传算法原理及其应用. 北京: 国防工业出版社, 1999)

HE Yuan, born in 1977, Ph. D. candidate, instructor. His research interests include computer information security and vulnerability discovery.

ZHANG Yu-Qing, born in 1966, Ph. D., professor. His research interests include network and information system security and vulnerability discovery.

ZHANG Guang-Hua, born in 1979, Ph. D., associate professor. His research interests include trust compute, wireless network security.

Background

This paper mainly researches on applying the genetic algorithm to the fuzz test of vulnerability discovery in android device driver. By now, there are many vulnerability discovery techniques, such as static analysis, dynamic taint analysis. Unfortunately, these tools are not suit to discover vulnerability in device driver. Because these technologies can't solve the communication between driver and device. Symbolic execution leverages the way of virtualization hardware solve the problem successfully. But there is no Symbolic system of android at present.

Traditional fuzz test is commonly used to discover vulnerability. But its disadvantage is that has no knowledge of the target, so it is inefficiency when it product test cases. So it hardly discovers any vulnerability if we don't improve the technology. After analysis the problem of traditional fuzz, we present a new genetic algorithm base on black-box test to produce test case. Different from the existing genetic algorithm, we take advantage of the execution results to generate the next generation of test case with error message, and no need the support of the source code. We preserved valid parameters meanwhile change invalid parameters by genetic algorithm. Algorithm leverage the execution result improves the convergence rate of the test cases, make the validity of the test cases is greatly increased. But vulnerability discovery not only need to perform function calls correctly, you also need to mine the unknown vulnerabilities, so in the genetic algorithm executes, needs to retain a certain randomness. For certain types of vulnerabilities, we can modify the parameters of the corresponding test case so it can be tested. Through the improved, test case composes from the original single random test parameters to the effective parameters, default parameters, random parameter. It not only improved the effectiveness of the test cases but also improved the ability to discover vulnerability.

At last we present the system called add-fuzz of vulnerability discovery in android device driver based on improved genetic algorithm. We performance the add-fuzz on many popular Android phones with different versions, and 9 security vulnerabilities were detected. The results demonstrate the effectiveness of the proposed method.

This paper is mainly supported by Projects of National Natural Science Foundation of China No. 61572460 and No. 61272481. Supported by the Open Project Program of Beijing Key Laboratory of IOT Information Security. This paper detected many vulnerabilities in android device driver, which is a very important part of that project.