计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20190655
56(10): 2097－2111, 2019

安全漏洞自动利用综述

赵尚儒[1,2] 李学俊[1] 方 越[1,2] 余媛萍[3,5] 黄伟豪[4,5] 陈 恺[4,5] 苏璞睿[3,5] 张玉清[1,2]
[1](西安电子科技大学网络与信息安全学院 西安 710071)
[2](国家计算机网络入侵防范中心(中国科学院大学) 北京 101408)
[3](中国科学院软件研究所可信计算与信息保障实验室 北京 100190)
[4](信息安全国家重点实验室(中国科学院信息工程研究所) 北京 100195)
[5](中国科学院大学 北京 100190)
(zhaosr@nipc.org.cn)

A Survey on Automated Exploit Generation

Zhao Shangru[1,2], Li Xuejun[1], Fang Yue[1,2], Yu Yuanping[3,5], Huang Weihao[4,5], Chen Kai[4,5], Su Purui[3,5], and Zhang Yuqing[1,2]
[1](*School of Cyber Engineering, Xidian University, Xi'an* 710071)
[2](*National Computer Network Intrusion Protection Center (University of Chinese Academy of Sciences), Beijing* 101408)
[3](*Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)
[4](*State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences), Beijing* 100195)
[5](*University of Chinese Academy of Sciences, Beijing* 100190)

Abstract With the increase of security vulnerabilities, it has been a considerable challenge to evaluate and repair vulnerabilities efficiently. However, the current assessment of the availability of vulnerabilities mainly depends on manual methods. How to intelligently and automatically exploit security exploits is a hot research issue in this field. In this paper, the literature on automated exploit generation of security vulnerabilities from 2006 to the present are investigated. We analysize current research progress, point out the development trend of exploit generation research, and summarize the general framework of automated exploit generation of vulnerabilities. We sort out the current research results from the three aspects of information input, vulnerability types and utilization methods, and discuss the effects of the three aspects on the automated exploit generation of vulnerabilities. Then the current shortcomings and challenges of automatic exploit generation of vulnerabilities are analyzed, and the future research trends and directions are also pointed out.

Key words vulnerability exploitation; exploit generation; automatic generation; vulnerability; automatic exploit

摘 要 随着安全漏洞数量急剧上升,高效率地评估与修复漏洞面临更大的挑战.目前漏洞的可利用性评估主要依赖人工方法,如何智能化和自动化地进行安全漏洞利用是本领域一个热点研究问题.调研了

收稿日期:2019-06-11;**修回日期**:2019-09-11
基金项目:国家自然科学基金项目(U1836210,U1836211)
This work was supported by the National Natural Science Foundation of China (U1836210, U1836211).
通信作者:张玉清(zhangyq@nipc.org.cn)

2006 年至今安全漏洞自动利用文献,分析了现状并指出了漏洞利用研究的发展趋势,同时给出了漏洞自动利用的一般框架;分别从漏洞自动利用的信息输入、漏洞类型和利用方法这 3 个角度对当前研究成果进行了梳理,指出了这 3 个角度对漏洞自动利用的影响;分析了漏洞自动利用研究的不足与挑战,并对将来的研究趋势进行了展望.

关键词 漏洞利用;利用生成;自动生成;安全漏洞;自动利用

中图法分类号 TP391

随着互联网行业的发展及软件系统复杂性增加,潜在的安全漏洞风险也在随之上升.与此同时各大企业、政府、高校逐渐重视安全,针对安全的相关投入也在逐渐增多.相应地,漏洞发现与报告也逐年增加,图 1 统计了美国国家漏洞数据库(National Vulnerability Database, NVD)披露的历年漏洞记录数量[1],仅 2019 年上半年(截至 7 月 1 日)披露的漏洞数量就超过了 2016 全年披露漏洞数量.然而已经披露的漏洞并不意味着该漏洞已经完全被修复,2017 年 5 月 12 日,WannaCry 勒索软件爆发就说明了这一点.它利用了 2017 年 3 月就已经发布了补丁的 Eternal-Blue 漏洞攻击脆弱系统.据了解,攻击者有平均 30 天[2]的时间利用公开的补丁信息来恢复漏洞信息,并攻击尚未修补的系统.

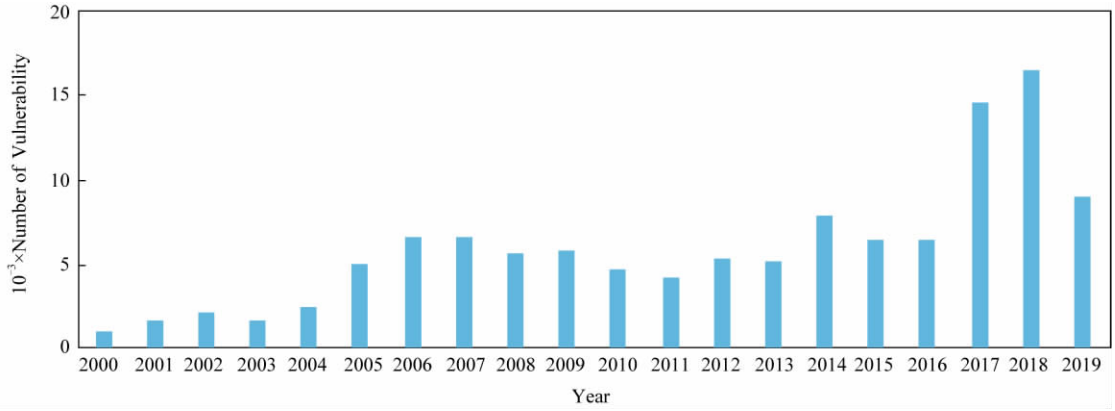

Fig. 1 NVD disclosed the number of vulnerabilities over the years

图 1 NVD 历年披露漏洞数量

当前的漏洞分析和修复工作仍然存在不少难题.首先,软件供应商和研究人员难以确定所提交漏洞的可复现性[3].漏洞复现本就是一项需要耗费时间的工作,而由于提交的漏洞报告的质量参差不齐,不少漏洞信息更是存在不准确、不完整的问题.安全研究人员可能还需要大量的时间来找出安全漏洞报告中存在的错误,以及补全缺失的信息.这些都严重拖慢了漏洞修复的速度,甚至会因为复现失败对漏洞造成误判.另一方面,目前对漏洞的可利用性判断也存在困难.从漏洞复现到漏洞利用,这个过程要对二进制程序以及程序源代码的运行过程有很透彻的分析,往往需要耗费安全研究人员大量的心血去研究.由于安全漏洞修复的周期长、资源有限,软件供应商往往需要快速判断漏洞的危险性以分配资源.因此,如何快速分析、评估漏洞的可利用性是当前漏洞发掘与分析的关键问题之一[4-6].

虽然软件自动漏洞利用技术已经取得了初步进展,但是软件复杂性的增加、控制流完整性检测和软件漏洞类型的多样性,给漏洞利用评估带来了更多的挑战[7].如何进一步探索和研究软件漏洞,提出更加高效可靠的自动化解决方案已经成为当前热门的研究领域.为此,我们调研了 IEEE, ACM, Springer 等出版期刊以及安全会议中有关漏洞利用的文献,尝试总结漏洞自动利用目前的研究成果,指出该领域的研究方向.图 2 展示了近年来漏洞自动利用文献数量,可以看出漏洞自动利用相关研究数量正在稳步上升,该领域逐渐受到关注,相信更多的研究成果即将出现.

本文的主要贡献有 3 个方面:

1) 系统地调研了 2006 年至今的近 70 篇漏洞自动利用文献,指出了漏洞利用研究的发展趋势,总结了漏洞自动利用的一般框架;

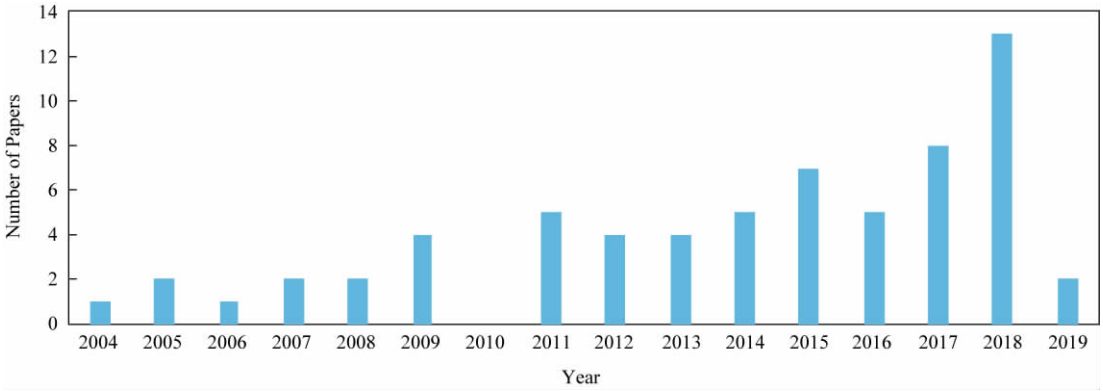

Fig. 2 Literature number of automatic exploitation generation

图 2 漏洞自动利用文献数量

2) 首次从漏洞利用的信息输入、漏洞利用的漏洞类型和自动利用的关键方法这 3 个角度对漏洞利用进行梳理,并分别指出了各角度中相关方案优缺点;

3) 分析了当前漏洞自动利用研究中的不足,总结了面临的七大机遇与挑战,并指出了未来的研究趋势与下一步研究方向.

1 漏洞自动利用

1.1 漏洞利用的定义

漏洞(vulnerability)是信息技术、信息产品、信息系统在需求、设计、实现、配置运行等过程中,有意无意之间产生的缺陷(bug).这些缺陷一旦被恶意主体利用,可能会影响构建在信息系统之上正常服务的运行,对信息系统的机密性、完整性以及可用性造成严重损害[8].然而一些漏洞可能不是系统安全的威胁,而只会导致程序在输出数据时出现特定的错误输出,也就意味着该漏洞是不可利用的.仅当漏洞可以被利用时才被称为安全漏洞,攻击者可以使用该漏洞程序执行恶意行为并破坏计算机安全[9].

漏洞利用(exploit),本意为"利用",指的是使用程序中的某些漏洞来得到计算机的控制权(使自己编写的代码越过具有漏洞的程序的限制,从而获得运行权限).在英语中,本词也作名词,表示为了利用漏洞而编写的攻击程序,即漏洞利用程序[10].漏洞自动利用(或称漏洞利用代码自动生成)就是通过给定的数据(如可执行文件、源代码、补丁等),生成可以利用该漏洞达到某特定目的的数据或者代码的过程的自动化实现[11].一般来说漏洞利用生成的利用代码需要获得目标系统最高权限(例如拿到 shell),虽然部分工作暂时不能做到这一点,但是有朝着拿到 shell 方向努力或者为此方向做出了贡献.

1.2 漏洞自动利用的发展

与漏洞利用有关的研究非常广泛,主要包括:漏洞自动利用、攻击方法、防护绕过、代码生成以及内存布局等.

1) 攻击方法

由于漏洞利用本质上是各种攻击方法的实例化,所以攻击方法是漏洞利用的研究基础.从各种内存冲突方案[12-14]到数据流攻击[15-16],攻击方案的发展给漏洞利用带来新鲜的血液.

2) 防护绕过与代码生成

随着安全防护措施的普及,漏洞利用研究需要考虑这些问题.当前研究中防护绕过和代码生成往往是相辅相成的,这主要是因为防护绕过的办法大部分体现在输出代码上,而代码生成的时候也会涉及防护绕过的问题.除了普通的防护绕过方案[17-19],DOP[20-23],Shellcode[24-29]也是常见的解决方案.

3) 内存布局

由于内存漏洞研究广泛、危害大,所以有关内存布局的研究也是主要的方向之一,如何精确地控制内存数据以达到预期目的是它们的研究目标[30-33].

我们将近年来漏洞自动利用研究的主要文献进行了汇总,如图 3 所示.最早的漏洞自动利用尝试可以追溯到 2006 年,Yang 等人尝试使用符号执行生成文件系统挂载程序的恶意硬盘[34].随后,有关漏洞自动利用的研究逐渐发展,研究数量总体呈上升趋势,各研究主要围绕着 3 个方向展开:不同输入信息处理方案、漏洞类型以及自动利用方法.输入信息方面研究者尝试通过可执行文件、源码、补丁、漏洞报告[43]和 PoC 程序等数据不断尝试生成漏洞利用.

在漏洞类型上,最初研究以内存溢出及 Web 注入为主,近年来,也有关于安卓恶意消息、UAF、内存读取等漏洞类型的利用研究.在自动利用方法上,符号执行这一强大的方法自 2006 年的 EXE 至 2018 年的 FUZE 仍然受到研究者的喜爱.与此同时,为了获得更好的效果摆脱已有工具的限制,大家也不断尝试新的方法:模糊测试、污点分析、模型检测、形式逻辑、自然语言处理等技术也纷纷得到应用.

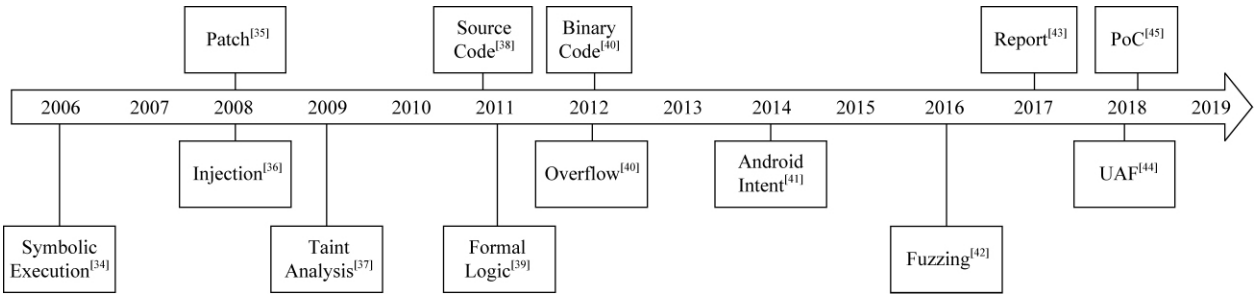

Fig. 3 The development of automatic exploitation generation of vulnerabilities

图 3 漏洞自动利用的发展

1.3 漏洞自动利用框架

通过调研现有文献,我们给出了漏洞自动利用一般过程,包括信息提取、漏洞识别、路径发现、状态求解及代码生成,如图 4 所示.总的来说,漏洞类型将会指导整个漏洞利用过程.漏洞利用首先从可执行文件、源码等输入数据中提取利用需要的信息,利用路径发现与状态求解以获取利用案例,最后辅以代码生成技术,生成漏洞利用的程序或者数据.

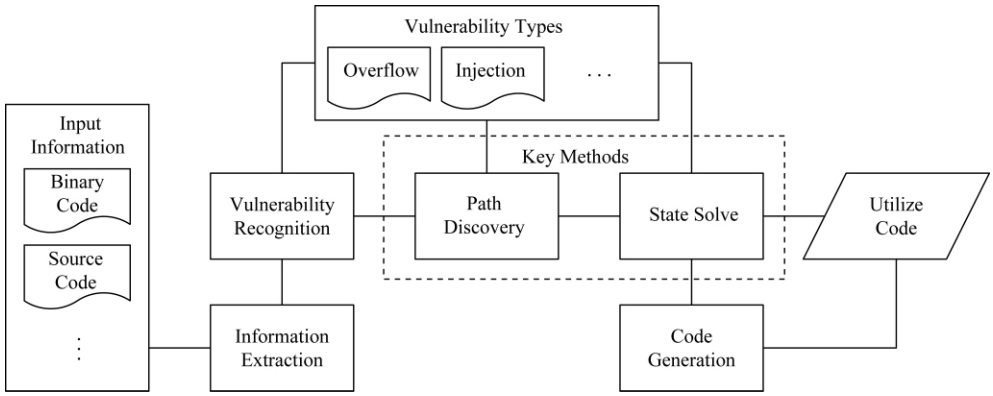

Fig. 4 Vulnerability automatic exploitation generation framework

图 4 漏洞自动利用框架

具体来说,首先从输入数据中提取信息.不同于漏洞挖掘,漏洞利用可以在已知存在漏洞的情况下生成对应漏洞的利用数据,所以给出的数据类型可能十分多样,同时也可能包含非常丰富的信息.例如给出的可能是存在漏洞的可执行文件、源代码这种通用数据,也可能是补丁、PoC 这种针对个别漏洞的专用信息.如何对上述类型各异的信息进行处理,并从中提取出可对后续漏洞利用进行有效指导的信息是本步骤的研究重点.当然漏洞利用也可以结合漏洞挖掘,在未知漏洞的情况下先行挖掘漏洞,再对挖掘出来的漏洞进行利用.这种情况对利用框架的后续架构没有太大影响.由于给定的信息可能不足以支持漏洞利用,接下来需要基于提取到的信息获取出漏洞类型及其他相关利用信息.这一步是非常灵活的,如果给定的信息可能包含足够的漏洞信息,甚至可以跳过此识别的步骤.

根据漏洞信息尝试发现利用路径,求解出可用的利用状态与输入的关系.这一部分可以说是漏洞利用的关键,因为这一步可以发现利用状态与输入的关系,即意味着将来可以根据此关系推导出利用目的对应的输入,而这正是达成漏洞利用最终目的的关键操作.最后,应用代码生成技术,生成符合特殊要求的代码或者数据,例如符合特定规则检测的数据、同一漏洞的不同利用实例等.

2 漏洞利用的信息输入

在漏洞利用的过程中我们往往需要从一定的数据来源中获取漏洞利用信息.我们统计了当前研究中使用信息输入情况,如图 5 所示,可以看出,直接使用可执行文件进行漏洞利用的研究所占比例最多,占 66%,源码的利用研究数量次之,占 29%,另外还有 5%将使用其他的信息输入例如补丁、PoC 等.与其他类型的信息相比,可执行文件最容易获取,而源码、补丁等数据则面临不被公开、不存在等的问题.然而随着开源软件的流行,基于源码漏洞利用也备受关注.源码相比可执行文件保留了更多的信息,但是不同的源码语法规则、执行逻辑的差别导致无法统一利用的问题仍然困扰着研究者.而其他类型的信息包括补丁、PoC 等则包含了更为精炼的漏洞信息,甚至直接指明了利用路径,但是这类信息数量少、质量参差不齐,严重降低了利用效率.

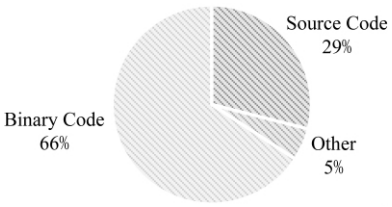

Fig. 5 Information input for exploits

图 5 漏洞利用的信息输入

本节将进一步讨论不同的信息输入对漏洞自动利用的影响.

2.1 可执行文件

可执行文件是指源代码经过预处理、编译、汇编、链接等步骤以后,以特定结构组织的二进制指令序列文件.可执行文件有利用时需要的程序运行级别的细节,例如堆栈帧、内存地址、变量放置和分配.但可执行文件在分析时的扩展性更差,且不具有源码级别的抽象信息[46].

由于可执行文件是一系列可执行的二进制指令,根据是否将指令输入硬件执行,我们可以将可执行文件分析方法分为静态分析和动态分析.

静态二进制分析过程通常从加载和处理要分析的二进制文件开始.处理步骤包括解析二进制文件、生成二进制文件的汇编指令的中间语言表示以及构建控制流图.虽然这项技术提供了一个系统检查所有可能的程序路径的能力,但是静态二进制分析可能很慢,并且在处理间接跳转语句时有局限性[47].一般来说,静态分析使用切片算法和关于接收输入数据的点和异常终止点的信息,选择仅由处理输入数据的指令组成的子空间.对于选定的子跟踪,将构造路径谓词,生成用户指定的利用代码[48].由于可执行文件可读性差,各架构格式不统一,还需要将二进制代码转换为中间文件再进行分析.

文献[49]首先使用 IDA Pro 将二进制代码解包,并使用 BinNavi 翻译成 REIL 中间表示.接着输入从输入函数到潜在可利用语句的 REIL 指令序列.最后尝试寻找一个执行与潜在可利用语句控制流所需的一组输入约束.

文献[50]首先分解 EVM 字节码并重建控制流图(control flow graph, CFG).接下来,扫描该 CFG 以获得关键指令以及状态改变指令.接着探索从 CFG 引导的根到这些指令的路径,通过符号执行从这些指令创建一组路径约束.最后利用生成模块解决了关键路径和状态改变路径的组合约束,以生成利用.文献[51]反汇编二进制代码得到程序的中间表示(intermediate representation, IR)和 CFG,并以此检查在漏洞利用过程中使用的程序信息,例如二进制代码中是否存在汇编代码"JMP ESP"和字符串"SH".接着使用符号执行来搜索程序的路径,并根据二进制代码的控制流图挖掘漏洞.最后绕过系统保护自动生成利用代码.

文献[52]从执行跟踪中构建一个信息和控制流混合图(HI-CFG),以结合代码、数据以及它们之间关系的信息.在获取 HI-CFG 后,使用符号探索来寻找导致故障的缓冲区内容或者导致后续缓冲区的期望故障触发内容.符号探索通过构造输出值和路径条件的符号表达式,并使用决策过程求解这些公式,生成程序一部分的可行执行路径.

文献[40]将 x86 汇编转换为适合符号执行的中间语言.具体来说,对于执行的每一条指令,符号执行器都会将指令 JIT 编译成 BAP 中间语言.然后直接在 IL 上执行符号执行,引入与特定攻击有效载荷相关的附加约束,并将公式发送给 SMT 求解器以检查可满足性.

Coppelia[53]首先使用 Verilator 将 RTL Verilog 翻译成逻辑等价的 C++代码,然后将安全关键断言添加到生成测试平台,并使用 Clang 编译器将新翻译的设计编译成 LLVM 字节码,最后定义违反安全关键断言的处理器状态,使用符号执行找到输入将系统从初始状态带到违反状态的路径.

动态分析常见的方法有模糊测试和动态污点分析[54].它们尝试直接在硬件中执行或者模拟执行该二进制指令序列,在执行的过程中定位、提取利用需要的信息,例如堆栈数据、内存变量地址等.这就意味着需要考虑如何获取寄存器及内存的信息.另外,为了解决设备外执行的问题,最近还提出了模拟执行的方法.

文献[55]允许研究人员开始在目标设备上执行固件.开始时固件在设备上执行,直到到达断点,接近研究者希望更仔细分析的有趣代码段.这也允许研究人员快速到达感兴趣的代码段,而不必从绝对开始就模拟固件.当到达断点时,在目标设备上停止执行,并执行文本转换.设备的整个状态(内存和寄存器)被传输到仿真器,仿真器然后从设备停止的地方恢复执行.上下文切换可以发生多次,每当这种情况发生时,仿真器拥有的与设备不同步的任何状态(如存储器和寄存器)被复制到目标设备,然后从模拟器停止的地方继续执行.

文献[56]首先指定目标设备的硬件,创建一个与目标处理器非常相似的虚拟机,根据目标设备的布局人工映射内存范围.随后将把涉及这些区域之外的地址的任何操作转发给目标设备.如果代码和内存没有映射,那么所有的内存操作都将被转发到目标设备.过于具体而无法放入框架的定制功能,例如呼叫监视器、存储器和寄存器状态转移功能也在虚拟配置文件中实现.

2.2 源代码

源代码(source code)是指一系列人类可读的计算机语言指令,最终目的是将人类可读的文本通过编译器翻译成为计算机可以执行的二进制指令.我们所说的源代码通常指的是C/C++,BASIC,C#,JAVA,PASCAL等高级语言编写的代码.

源代码包含抽象信息,如变量、缓冲区、函数和用户构造的类型,使自动推理更容易并具有更好的伸缩性[38].但同时高级语言的抽象隐藏了程序操作的细节,而这些细节对于检测错误和评估它们的严重性非常重要[48].

源代码作为编程语言,最大的特点就是可以被译成可执行文件.这也就是说,可以在转换之后直接使用可执行文件的分析方法.另外,源代码保留的抽象信息可以帮助我们进行静态分析,获取漏洞利用的有关信息.

AEG[38]分析源代码程序,生成符号执行公式并求解它们,执行二进制分析,生成二进制运行时约束,并将输出格式化为可以直接输入易受攻击程序的实际利用字符串.NAVEX[57]给定应用程序源代码,识别应用程序和相应模块中易受攻击的接收器.在这个阶段分别分析每个模块,并且只优先考虑那些有漏洞的模块从而显著减少搜索空间并有助于提高可伸缩性.

文献[58]使用源代码弱点白盒检测引擎和开源收集引擎将分析目标的源代码转换为中间代码,并对转换后的中间代码和处理后的数据进行编码,然后将其传输到弱点分析引擎服务器.其利用基于编译器的符号执行在源代码级进行弱演绎,提取测试结果的弱代码连接黑盒测试.

除此之外,由于部分高级语言为解释语言,它们可以通过修改解释器来进行符号执行.文献[59]构建了一个基于Java符号执行框架,可以理解为一个非标准的Java字节码解释器.它在解释代码时强制执行路径探索;例如当解释if语句时,创建2个程序状态,以便探索2个分支.该解释器提供了一组路径选择策略,可以从各种约束求解器中进行路径条件选择.

2.3 其他信息

除2.1~2.2节提到的可执行文件以及源代码之外,还有补丁、PoC等信息可以利用,这一类信息数量更加少.它们的特点是具有超高的信息密度,它们往往直接提示漏洞位置、触发函数等等关键信息.但是这可能在精简的过程中遗漏了某些信息而导致利用失败.所以大部分研究会同时结合可执行文件或者源代码等其他信息,在保证利用成功的同时提高利用效率.

1) 补丁文件

APEG[35]使用二进制分析工具将未安装补丁的程序与已安装补丁的程序进行对比,识别出添加检查的差异位置,检查条件等.根据识别出来的检查与可执行文件生成到达检查点以前的约束公式,接着自动生成不能通过添加的检查的输入.

2) PoC

FUZE[44]首先将一个PoC程序作为输入,识别提取漏洞利用需要的关键信息,例如:①易受攻击对象的基址和大小;②与留下悬空指针的自由站点相关的程序语句;③与悬空指针取消引用的站点相对应的程序语句.然后,它利用内核模糊化来探索各种系统调用,从而改变内核异常的上下文.在每个包含不同内核崩溃的上下文中,FUZE进一步执行符号执行,跟踪评估可能对漏洞利用有用的原语.

Revery[45]提出一种使用提供的 PoC 输入测试目标应用程序,基于已知崩溃路径相关的信息,结合堆布局导向模糊测试技术,获取漏洞程序更多可利用状态的方法.它跟踪每个指针和内存对象的状态,并捕获崩溃路径上的安全信息、运行时信息等.Revery 还检查损坏的内存对象(表示为异常对象)以及可用于定位异常对象的对象,从路径中检索布局贡献者信息,创建它们之间的点对点关系.基于这些指令和对象,Revery 获得了一个布局贡献者有向图来描述漏洞的内存状态和贡献者.

3) 异常输入

PolyAEG[60]接受一个易受攻击的程序和一个异常输入.它使用给定的异常输入动态运行易受攻击的程序,这些异常输入可以使程序崩溃,跟踪每个指令并执行动态污点分析以收集执行信息.接着分析污点传播过程来检测控制流的劫持点,并提取污染的存储区域来存储所使用的跳转指令和 Shellcode.基于污点执行信息生成路径约束,确保当程序以漏洞为输入运行时劫持点是可到达的.最后利用指令构建一个跳转指令链,将程序的执行重定向到 Shellcode.至此,通过前面阶段中确定的指定数据依赖关系和路径约束修改相关输入字节来生成一个漏洞利用.

2.4 小 结

第 2 节介绍了漏洞利用中常见的输入信息.其中可执行文件最为研究者青睐,因其使用广泛、易于获取.当然其他信息例如源代码、PoC 等虽然各有优缺点,但是其中蕴含的信息很可能对漏洞利用十分关键.如果漏洞利用的过程中可以获取的信息越多,漏洞利用的成功率将会越高.因此探索新的信息来源、结合多种信息利用是当前研究热门话题.

3 漏洞利用中的漏洞类型

漏洞利用和目标漏洞类型息息相关.我们统计了当前漏洞自动利用研究中漏洞类型的情况,如图 6 所示.可以看出:在自动利用漏洞类型中,内存溢出类漏洞最多,占 55%,Web 漏洞其次占 24%,另外还有 21%的研究尝试其他的漏洞类型.内存溢出漏洞是最为经典的漏洞,关于漏洞的研究的起始正是从内存溢出开始的.而且内存漏洞与其他漏洞相比,更容易带来简单而严重的危害,所以大部分研究者会考虑从内存类漏洞进行研究.紧接着就是 Web 注入类漏洞,Web 是当前最流行的开放系统和信息获取渠道,自 20 世纪 80 年代以来,Web 应用程序已从文本,图像和超链接的静态 HTML 页面演变为可自定义和交互式页面.这导致了该类漏洞产生,也促进了对 Web 漏洞的研究[61].与此同时,其他漏洞类型也同样有研究的价值.

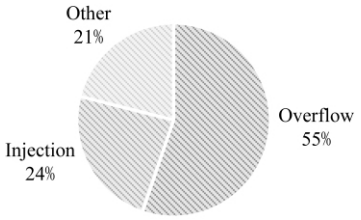

Fig. 6 Vulnerability type of exploit

图 6 漏洞利用的漏洞类型

3.1 内存溢出类漏洞

内存溢出类漏洞主要包括:栈溢出、堆溢出和格式化字符串.内存溢出漏洞之所以受到大家的关注,主要是因其危害性严重.一般来说,当前计算机运行程序,首先要将程序代码、数据复制到内存,然后再交由 CPU 执行.如果程序执行过程中出现内存意外读写,就有可能影响到运行的代码或者数据.若该漏洞被攻击者所利用,就会导致各种可能的危害,甚至是被执行任意代码,也就意味着该计算机被攻击者完全控制.

内存类漏洞由于发生在内存空间,因此非常需要获取程序运行时的内存信息.因此大部分研究需要针对如何获取漏洞触发时信息作出讨论.

符号程序计数器(x86 机器中的 EIP 寄存器)包含下一条要执行的指令的地址,所以控制寄存器是所有控制劫持攻击的一个常见的攻击目标.因此,Crax[62]提出监测 EIP 寄存器的状态是解决不同类型的控制流劫持漏洞的一种全面而简单的方法.当符号执行探索路径并污染内存时,使用符号数据更新 EIP 寄存器,从而触发漏洞利用.漏洞利用生成将搜索内存以找到可用的内存区域来注入 Shellcode 和 NOP sled,并将 EIP 寄存器重定向到 Shellcode.

除了 EIP 寄存器之外,损坏的指针可能间接改变控制流.特别是,分配给符号指针的符号数据意味着可以将任意数据写入任意地址.当检测到符号写入时,写入操作的目标将被重定向到敏感数据,如返回地址,.dtors 或者 GOT,以间接更新 EIP 寄存器[46].

对于堆溢出,如果攻击者指定的输入是符号化的,而且关键操作最终会操纵符号字节,那么攻击者

的输入将可能到达某些限制下的关键操作.这些约束决定了攻击者对这些关键操作中使用的值进行控制的级别.因此,一旦检测到目的地的符号数据流,就可以发现堆利用原语[63].

除了直接检测内存数据以外,Pangr[64]尝试了新的基于行为的建模方法.分别对格式字符串、栈溢出和堆溢出漏洞的行为进行建模,并利用漏洞在符号执行过程中触发的内在语义来寻找有价值的漏洞.在找到有利于以后漏洞利用的易受攻击点后,漏洞分析器记录输入值和上下文信息,如寄存器、堆栈、堆和环境变量等.

3.2 Web 注入类漏洞

Web 注入类漏洞主要包括:SQL 注入(SQLi)、XML 注入(XMLi)和跨站脚本(XSS).3 种类型的漏洞本质上是相似的.例如 SQLi 和 XSS 都依赖于将恶意内容注入到合法数据并从输入源流动到漏洞触发点.当恶意内容通过查询注入数据库或当它到达向客户端发送内容(即代码)的类似回显的语句时,漏洞即被触发[65].

直观地,我们可以使用测试框架直接测试后端网络服务.文献[66]介绍 SOLMI,一个用于 XML 注入的基于求解器和变异的测试生成框架.它使用一组变异操作符,可以操纵非恶意的 XML 消息生成 4 种类型的 XMLi 攻击来绕过 XML 网关并针对后端网络服务进行攻击.

文献[36]尝试检测以下 4 种模式来识别漏洞:1)不受信任的数据是从一些污点源读入的,例如用户控制的文件、网址请求、cookie 值或网络源.它随后可以存储在任意对象中,并作为参数或返回的结果传入传出.2)有些方法可能会从旧对象中派生出新对象.其中一些,如果传递给一个不可信的对象,将产生一个不可信的对象.3)任何不受信任的数据,无论是来自原始污点源还是通过传播程序导出的,都不能用于任何污点接收器,例如数据库访问例程.4)如果对象已经通过引用或转义对象内容的清理程序,则前面的规则不适用.

文献[37]比较由分别在 2 个输入上执行的 PHP 程序发布的数据库状态(例如选择、插入).检查器比较第 1 对相应的语句,然后比较第 2 对语句,依此类推.如果任意对中的语句都是有效的 SQL,但具有不同的语法结构(即解析树),则检查器发出攻击信号.

NAVEX[57]构建了一个攻击字典,用于实例化针对每类漏洞的分析模板.它包含:1)接收器.这些指令执行攻击的恶意内容.例如回显和打印 PHP 函数是 XSS 攻击的接收器;2)清理器.包括一个广泛的 PHP 清理列表,包括内置的清理函数和操作符,它们可以隐式清理输入(例如转换操作符).3)遍历类型.它指定了图形所需的遍历类型.4)攻击字符串.攻击字符串是可能出现在接收器上的(恶意)值的规范.目前,攻击字典包含 SQLi,XSS、文件包含、命令注入、代码执行和 EAR 的条目.

符号执行也可以被用来识别 Web 注入类漏洞.Craxweb[67]将准备好的符号数据注入到一个超文本传输协议请求中.如果符号数据可以在符号执行过程中通过套接字传播到 HTTP 响应或数据库查询,则表明响应或查询易受攻击,可以由原始输入的符号数据控制.

3.3 其他类型漏洞

除了内存溢出和 Web 注入 2 类常见漏洞外,仍有许多类型的漏洞值得探索.

1) UAF 漏洞

对于栈或堆溢出漏洞,简单地改变 PoC 程序的上下文就可能促进对这些漏洞的利用,而 UAF 漏洞的利用需要对易受攻击的对象进行暂时和临时的控制,在这些限制下,上下文的微小变化通常不利于可利用性的探索.FUZE[44]首先准确定位悬空指针出现的位置,以及指针被 PoC 程序中定义的系统调用取消引用的位置.然后在上下文中执行符号执行,目的是确定上下文是否可以将内核执行导向可利用的机器状态.接着基于通过内核模糊化获得的内容来设置符号执行.最后通过使用符号执行来识别对开发真正有用的机器状态.

2) 参数篡改漏洞

当服务器端参数验证弱于客户端验证时,网络应用程序被认为是易受攻击的.换句话说,服务器对客户端提供的输入的格式执行的检查比客户端少.WAPTEC[39]利用网络应用程序中客户机中代码构成的关于参数验证检查意图的描述,直接从客户机代码中提取一个规范,然后用该规范检查服务器端代码的漏洞.

3) 安卓意图(intent)欺骗

意图是安卓系统中一个要执行的操作的抽象描述.意图消息要么携带特定目的地的信息,要么表达需要由某个能够管理它的流程提供服务的通用请求,还可以作为广播消息,通知一组感兴趣的进程发生了一些状态更改[41].然而,安卓意图传递机制不会向接收组件提供任何关于意图来源的信息,因此有

助于创建带有恶意输入数据的欺骗意图.如果这种恶意输入在被处理之前没有被应用程序正确验证或净化,则可能导致拒绝服务或者跨应用脚本执行[68].为了识别潜在的进程间拒绝服务攻击,文献[69]检查意图属性(即意图动作、额外数据或类别)的每次使用,并沿着相应属性的使用定义链执行反向数据流分析,以确定是否有可能导致未处理的空指针异常.对于跨应用脚本执行,文献[69]首先通过在应用程序中识别 WebView.loadUrl(…)的调用来识别易受跨应用程序脚本攻击的语句.从这样的语句开始,沿着传递给该语句的调用 WebView.loadUrl(…)的参数的 use-def 链执行向后数据流分析.如果这些参数中的任何一个是定义语句的使用,其右侧涉及提取意图属性,则认为该语句易受该漏洞的攻击,可以依据此路径生成漏洞利用.

4) 安全策略执行不一致

安卓框架利用基于权限的安全模型,提供对各种系统资源的受控访问.但是,敏感操作可能从不同的路径到达,这可能会导致安全检查失效.因此,权限不足的攻击者可能会通过采用缺乏安全检查的路径来执行敏感操作.Centaur[59]首先找到所有到达敏感操作的可行路径,然后给出每个可行路径所需的许可(所需的许可包含在每个路径条件中),接着验证可行路径之间的许可一致性,最后尝试生成使用可行路径验证可疑漏洞的输入.

5) 信息泄露

文献[15]提出了一种在面向对象程序中自动生成信息流泄漏漏洞的方法.他们的方法将自组合和符号执行结合起来,为给定的信息流策略和程序位置的安全级别规范组成一个不安全公式.不安全公式产生了一个模型,该模型用于为该漏洞生成输入数据.

3.4 小 结

第 3 节介绍了不同漏洞类型的特点,并对常见的内存溢出类漏洞、Web 注入类漏洞及其他漏洞类型的常见利用方式作总结.可以看出虽然不同种类的漏洞有所相似,但是进行利用甚至自动化利用时仍然有不可逾越的区别.另外,当前漏洞自动利用研究中,每次利用方案生成往往只能有一种漏洞类型,如何综合多个漏洞(同类型或者不同类型)利用仍然是有待研究的问题.

4 自动利用的关键方法

获得漏洞利用的相关信息以后,需要对这些信息进行处理,以得到漏洞利用中最关键的信息:漏洞触发与输入的关系.只有获得了这个信息,我们才有可能做到“利用”.常见的方法有:符号执行、模糊测试、污点分析等.由于这些方法都有各自的局限,当前研究主要针对这些方法进行改进优化.

4.1 符号执行

符号执行(symbolic execution)是一种程序分析技术,其可以通过分析程序来得到让特定代码区域执行的输入.目标程序的输入被当成是符号变量,当代码执行时,数据被“替代”为条件表达式和其他操作,结果递增地表示为对输入值的约束,以便在给定路径上继续执行.每次代码执行包含符号值的条件检查时,都会需要分叉执行,在真实路径上添加分支条件持有的约束,而在错误路径上添加它不持有的约束.最后通过使用约束求解器来查找满足约束的具体值,从而为程序生成测试用例[70].符号执行是自动利用中使用得最多的方法.自 2006 年有人首次使用[34]至今最新的研究[71],符号执行一直是漏洞利用中强有力的工具.

符号执行的弱点是执行过程中的路径爆炸问题,这给大规模网络应用程序上的漏洞生成带来了挑战.为应对这个问题,最基础的方法就是减少开销.Craxweb[67]利用并发测试的优势尝试提高效率.文献[46]提出了一种基于路径选择优化、选择性符号输入和伪符号变量惰性赋值的自动匹配漏洞生成方法来处理符号指针.文献[69]通过从易受攻击的语句启动静态符号执行,减少了必须计算路径的空间,而不是从应用程序入口点开始到从这些入口点可访问的所有语句执行正向符号执行.这种修剪可以减少利用生成的计算时间.而 EOEDroid[72]则尝试使用选择性符号执行探索事件处理程序中的路径并收集路径约束.更具体地说,给定一个应用程序,其调用“选择性符号执行”来重复测试每个事件处理程序,直到遍历所有内部感兴趣的路径.有趣的路径由子模块“启发式生成”发现.当一个分支被标记为有趣时,不管条件语句是否是符号性的,EOEDroid 都会强制遍历这条路径.同时,构造并保存相应的路径约束.对于每一轮测试,子模块“分析沙箱”用于保护分析环境免受污染,并保持每一轮测试的独立性.

除此之外,AEG[38]提出了一种针对更有可能被利用的路径优先化技术和预处理符号执行技术.例如只探索具有最大输入长度的路径,或者与 HTTP GET 请求相关的路径.然后使用基于启发式方法优先级队列路径优先化技术,使得程序首先选择可能

更易被利用的路径.文献[59]提出了一种分阶段的具体执行到符号执行的技术,用于分析像安卓框架这样的中间件软件.它将初始化阶段作为整个系统的具体执行运行,然后从具体执行提供的执行上下文中的一个入口点方法开始执行符号执行.避免了由于复杂的初始化阶段导致的状态空间爆炸,同时为符号执行提供了上下文,使得输入变量的类型和值信息可用.

4.2 模糊测试

模糊测试(fuzzing)是一种软件测试技术.其核心思想是将自动或半自动生成的随机数据输入到一个程序中,并监视程序异常,以发现可能的程序错误.模糊测试常常用于检测软件或计算机系统的安全漏洞.典型模糊测试范例:测试生成、崩溃检测和测试缩减.首先变异并产生符合输入规范的数据.然后定制数据各元字段中可以修改的部分.在执行过程中,评估执行的测试用例是否会产生导致利用的影响.每当发现新的漏洞时,最小化相关操作,并生成一个只包含一组基本操作的 PoC 代码作为证明[73].

针对模糊测试存在效率低下的问题.文献[58]创建一个针对模糊化的软件数据模型,并自动对数据文件和软件本身进行分析.通过静态分析(弱点信息、输入文件结构)进行建模,建模后的数据结构和弱点代码通过连接模糊化测试,用于提取输入数据.Pangr[64]尝试使用符号辅助模糊化利用了符号执行对语义理解的优势,并且模糊化的执行时间短.Deepfuzz[42]提出一种结合了初始种子生成协同执行、路径概率分布、路径选择和约束模糊化的深度模糊算法.

另外,Revery[45]采用了一种面向内存布局的模糊化解决方案来扩展发散路径.它只探索内存布局与 PoC 输入相似的各种路径.因此,它能驱动模糊器探索接近内核崩溃的路径,但是没有使用完整的碰撞该路径,而是使用前面提到的布局贡献者指令作为模糊器的指导.而且,Revery 并不打算在模糊化过程中匹配确切的崩溃路径或触发漏洞.相反,它忽略了崩溃路径中的大部分指令,只保留前面提到的布局贡献者指令,这可能产生与漏洞类似的内存布局.因此,模糊器可以探索许多不同的路径,并有更好的机会找到可利用的状态,同时保持漏洞的记忆状态.Revery 使用启发式方法在不同的路径中寻找可利用的状态和劫持点,并试图合成新的利用输入,以触发分叉路径中的可利用状态和崩溃路径中的漏洞.它采用一种新的控制流拼接解决方案将分叉路径和碰撞路径拼接在一起,然后利用轻量级符号执行来生成利用输入.

4.3 污点分析

污点分析是一种跟踪并分析污点信息在程序中流动的技术.在漏洞分析中,使用污点分析技术将所感兴趣的数据(通常来自程序的外部输入)标记为污点数据,然后通过跟踪和污点数据相关的信息的流向,可以知道它们是否会影响某些关键的程序操作,进而挖掘或利用程序漏洞.

ARDILLA[37]介绍了一个使用污点分析识别 Web 注入漏洞的 5 项规则:

1) 污点源是输入.在测试中的 PHP 程序的执行过程中,污染源会导致污染数据.ARDILLA 为从输入参数中读取的每个值分配一个唯一的污点,由该值的来源标识.

2) 污点集描述了每个运行时值如何受到污点源的影响,并且可以包含任意数量的元素.

3) 污点传播指定运行时值如何获取和丢失污点.ARDILLA 通过应用程序代码中的分配和过程调用传播未更改的污点集.在调用不是污点过滤器的内置 PHP 函数时,ARDILLA 为返回值构造了一个污点集,它是函数参数值污点集的联合.ARDILLA 还通过合并组件字符串的污染集,为串联创建的字符串值构建污染集.在调用数据库函数时,ARDILLA 存储或检索数据值的污点.

4) 污点过滤器是内置的 PHP 函数,用于净化输入.在调用污点过滤器函数时,ARDILLA 为返回值创建了一个空污点集.ARDILLA 的用户可以选择指定污点过滤器列表.

5) 敏感的污点接收器是内置的 PHP 函数,可在 XSS 和 SQLi 攻击中利用.例如,XSS 的回显和打印以及 SQLi 的 mysql 查询.当到达对敏感接收器的调用时,ARDILLA 记录参数的污点集,指示从输入到接收器的数据流,从而指示攻击的可能性.

4.4 小 结

第 4 节介绍了漏洞自动利用中常见的 3 种方法.其中,符号执行可以精确获取输入输出与程序运行的关系,但是遇到状态空间爆炸的问题;模糊测试可以少量信息下尽可能地探索可利用状态,但是效率较低;污点分析可以快速获得输入输出存在的联系,但是不能精确解出状态与路径.当前研究主要围绕优化、结合这 3 种方法,但是目前效果仍不够理想,继续优化现有方法或是提出新的方法将是漏洞自动利用领域深远而困难的问题.

5 未来研究展望

在综述现有研究的同时,本文尝试总结漏洞自动利用领域当前面临的主要挑战,并结合相关研究进展给予展望.我们在表 1 中列举了漏洞自动利用中一些挑战与机遇.

Table 1 Top Seven Challenges and Opportunities

表 1 七大挑战与机遇

Challenge	Opportunity
Multiple Architectures	IoT Security Research
Unformatted Information	Natural Language Processing
Less Vulnerability Type	Research on Vulnerability
Multiple Vulnerability Synthesis	Deep Learning
State Space Explosion	Strong Computing Power
Path Selection Problem	Artificial Intelligence
Difficult Environment Modeling	Simulation Execution

1) 综合利用信息

在可执行文件的漏洞利用研究方面,当前主要集中在 X86 架构,然而随着物联网设备的普及,不同架构的设备与漏洞也随之增加.目前跨架构的信息处理仍然存在较大的困难.除了常见的可执行文件、源代码等信息,仍然有很多信息值得进行利用探索,例如漏洞库、博客、论坛等.当前不少研究在自动利用的过程中也尝试使用了自定义的信息,但尚未有系统性的总结与分析.当前研究漏洞种类单一,所使用的信息也较少,可以预见未来所需要的信息种类和数量将大大增加,因此需要对这些不常见的信息进行系统性分析与利用研究.综合多种信息进行漏洞利用看似是漏洞自动利用的完美解决方案.

2) 利用的漏洞类型与数量

漏洞自动利用由于受到技术以及精力的限制,过去研究涉及漏洞类型有限.然而随着漏洞自动利用的发展,相信将来会更多的漏洞类型被引入.同时我们也观察到,不同的漏洞类型也会有相似之处,甚至可以被一个统一的利用方法进行利用.那么,是不是所有的漏洞类型都可以被统一利用? 如果不行的话,不同类型的漏洞自动利用边界会在哪?

另外,在大部分实际情况中,如果要达到某种漏洞利用目的,往往需要同时结合多个漏洞来利用[74].但是当前的自动利用研究都无法产生综合多个漏洞的利用.这对于漏洞评估的实践来说仍是一个不得不重视的问题.

3) 利用关键方法的改进

当前利用方法仍然存在较大的局限.例如符号执行的状态空间爆炸问题、模拟执行的环境建模问题、模糊测试的效率低下问题等.符号执行使用至今已经有 10 余年的历史,经历了从最初的静态符号执行到动态符号执行、混合符号执行的发展.我们观察到,当前的研究也倾向于结合多种方法的优点以进行自动利用.虽然现有的工作已经有一部分的成果,但仍然有很大的改进空间.在将来,进一步优化、改进现有方法,提出更适合于自动利用的方法将是自动利用的核心问题.

6 总 结

随着计算机技术的不断发展,各种技术不断更新,漏洞自动利用作为漏洞评估的利器重新得到研究人员的重视.

本文总结了漏洞利用方面的最新研究,归纳了漏洞自动利用的整体框架,并从利用信息、漏洞类型、利用方法 3 方面对现有研究进行综述.

我们认为在未来:从更多的信息类型获取漏洞利用信息,整合多种信息进行漏洞利用将是发展趋势;拓展更多漏洞类型进行自动利用,多个漏洞同时综合利用将是研究重点;优化现有自动利用方法,提出新的漏洞自动利用方案将对漏洞自动研究、漏洞评估能力有重大影响.

参 考 文 献

[1] National Institute of Standards and Technology. National vulnerability database [EB/OL]. [2019-06-02]. https://nvd.nist.gov/

[2] Nappa A, Johnson R, Bilge L, et al. The attack of the clones: A study of the impact of shared code on vulnerability patching [C] //Proc of 2015 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 692-708

[3] Mu Dongliang, Cuevas A, Yang Limin, et al. Understanding the reproducibility of crowd-reported security vulnerabilities [C] //Proc of the 27th Security Symp. Berkeley, CA: USENIX Association, 2018: 919-936

[4] Miller C, Caballero J, Johnson N M, et al. Crash analysis with BitBlaze [J/OL]. [2019-06-10]. https://media.blackhat.com/bh-us-10/whitepapers/Miller/BlackHat-USA-2010-CMiller-Bitblaze-wp.pdf

[5] Heelan S. Automatic generation of control flow hijacking exploits for software vulnerabilities [D]. Oxford: University of Oxford, 2009

[6] He Liang, Su Purui. Research progress on automatic exploitation of software vulnerabilities [J]. China Education Network, 2016, (Z1): 46—48 (in Chinese)
(和亮, 苏璞睿. 软件漏洞自动利用研究进展[J]. 中国教育网络, 2016, (Z1): 46—48)

[7] Tan Tiantian, Wang Baosheng, Xu Zhou, et al. The new progress in the research of binary vulnerability exploits [G] //LNCS 11064: Cloud Computing and Security. Berlin: Springer, 2018: 277—286

[8] National Information Security Standardization Technical Committee (SAC/TC 260). GB/T28458—2012 specification for identification and description of security vulnerabilities [S]. Beijing: China Standards Publishing House, 2012 (in Chinese)
(全国信息安全标准化技术委员会(SAC/TC 260). GB/T28458—2012 安全漏洞标识与描述规范[S]. 北京: 中国标准出版社, 2012)

[9] Huang boyan, Huang shikun. Attack code to automatically generate interception control flow [D]. Taiwan, China: Taiwan Chiao Tung University, 2011 (in Chinese)
(黄博彦, 黄世昆. 自动产生拦截控制流程之攻击程式码[D]. 台湾, 中国: 台湾交通大学, 2011)

[10] Wikipedia. Exploit (computer security) [EB/OL]. [2019-06-02]. https://zh.wikipedia.org/wiki/%E6%BC%8F%E6%B4%9E%E5%88%A9%E7%94%A8

[11] Sun Hongyu, He Yuan, Wang Jice, et al. Application of artificial intelligence technology in security vulnerabilities [J]. Journal of Communications, 2018, 39 (8): 1—17 (in Chinese)
(孙鸿宇, 何远, 王基策, 等. 人工智能技术在安全漏洞领域的应用[J]. 通信学报, 2018, 39(8): 1—17)

[12] Koruyeh E M, Khasawneh K N, Song C, et al. Spectre returns! speculation attacks using the return stack buffer [C] //Proc of the 12th Workshop on Offensive Technologies (WOOT18). Berkeley, CA: USENIX Association, 2018

[13] Razavi K, Gras B, Bosman E, et al. Flip feng shui: Hammering a needle in the software stack [C] //Proc of the 25th Security Symp (Security 16). Berkeley, CA: USENIX Association, 2016: 1—18

[14] Xu Wen, Li Juanru, Shu Junliang, et al. From collision to exploitation: Unleashing use-after-free vulnerabilities in Linux kernel [C] //Proc of the 22nd ACM SIGSAC Conf on Computer and Communications Security (CCS'15). New York: ACM, 2015: 414—425

[15] Do Q H, Bubel R, Hähnle R. Exploit generation for information flow leaks in object-oriented programs [G] // LNCS 455: ICT Systems Security and Privacy Protection. Berlin: Springer, 2015: 401—415

[16] Hu H, Shinde S, Adrian S, et al. Data-oriented programming: On the expressiveness of non-control data attacks [C] //Proc of 2016 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2016: 969—986

[17] Kruegel C, Kirda E, Mutz D, et al. Automating mimicry attacks using static binary analysis [C] //Proc of the 14th USENIX Security Symp. Berkeley, CA: USENIX Association, 2005: 11—11

[18] Fogla P, Sharif M I, Perdisci R, et al. Polymorphic blending attacks. [C] //Proc of the 15th USENIX Security Symp. Berkeley, CA: USENIX Association, 2006: 241—256

[19] Hund R, Holz T, Freiling F C. Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms [C] // Proc of the 18th USENIX security Symp. Berkeley, CA: USENIX Association, 2009: 383—398

[20] Chen Ping, Xing Xiao, Mao Bing, et al. Automatic construction of jump-oriented programming shellcode (on the x86) [C] //Proc of the 6th ACM Symp on Information, Computer and Communications Security (ASIACCS'11). New York: ACM, 2011: No.20

[21] Schwartz E J, Avgerinos T, Brumley D. Q: Exploit hardening made easy. [C] //Proc of the 20th USENIX Security Symp. Berkeley, CA: USENIX Association, 2011: 25—41

[22] Homescu A, Stewart M, Larsen P, et al. Microgadgets: Size does matter in turing-complete return-oriented programming [C] //Proc of the 6th USENIX Conf on Offensive Technologies. Berkeley, CA: USENIX Association, 2012: 7—7

[23] Carlini N, Wagner D. ROP is still dangerous: Breaking modern defenses [C] //Proc of the 23rd Security Symp (Security 14). Berkeley, CA: USENIX Association, 2014: 385—399

[24] Arce I. The shellcode generation [J]. IEEE Security & Privacy Magazine, 2004, 2(5): 72—76

[25] Song Y, Locasto M E, Stavrou A, et al. On the infeasibility of modeling polymorphic shellcode [C] //Proc of the 14th ACM Conf on Computer and Communications Security. New York: ACM, 2007: 541—551

[26] Mason J, Small S, Monrose F, et al. English shellcode [C] // Proc of the 16th ACM Conf on Computer and Communications Security (CCS'09). New York: ACM, 2009: 524

[27] Prabhu P V, Song Y, Stolfo S J. Smashing the stack with hydra: The many heads of advanced polymorphic shellcode [J/OL]. [2019-09-10]. https://academiccommons.columbia.edu/doi/10.7916/D8RJ4R91

[28] Basu A, Mathuria A, Chowdary N. Automatic generation of compact alphanumeric shellcodes for x86 [G] //LNCS 8880: Information Systems Security. Berlin: Springer, 2014: 399—410

[29] Bao T, Wang Ruoyu, Shoshitaishvili Y, et al. Your exploit is mine: Automatic shellcode transplant for remote exploits [C] //Proc of 2017 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2017: 824—839

[30] Sotirov A. Heap feng shui in JavaScript [J/OL]. [2019-09-10]. http://www.mathcs.richmond.edu/~dszajda/research/summer_2014/papers/sotirov_heap_feng_shui_javascript_paper.pdf

[31] Παπαντωνίου I. Heap spray exploitation [OL]. [2019-06-10]. http://dione.lib.unipi.gr/xmlui/handle/unipi/5013

[32] Lu Kangjie, Walter M T, Pfaff D, et al. Unleashing use-before-initialization vulnerabilities in the Linux kernel using targeted stack spraying [C] //Proc of 2017 Network and Distributed System Security Symp. San Diego, CA: Internet Society, 2017

[33] Heelan S, Melham T, Kroening D. Automatic heap layout manipulation for exploitation [C] //Proc of the 27th Security Symp. Berkeley, CA: USENIX Association, 2018: 763—779

[34] Yang Junfeng, Can Sar, Twohey P, et al. Automatically generating malicious disks using symbolic execution [C] //Proc of 2006 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2006: 15—257

[35] Brumley D, Poosankam P, Song D, et al. Automatic patch-based exploit generation is possible: Techniques and implications [C] //Proc of 2008 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2008: 143—157

[36] Martin M C, Lam M S. Automatic generation of XSS and SQL injection attacks with goal-directed model checking. [C] //Proc of the 17th USENIX Security Symp. Berkeley, CA: USENIX Association, 2008: 31—44

[37] Kieyzun A, Guo P J, Jayaraman K, et al. Automatic creation of SQL injection and cross-site scripting attacks [C] //Proc of the 31st IEEE Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2009: 199—209

[38] Avgerinos T, Cha S K, Hao B L T, et al. AEG: Automatic exploit generation [J/OL]. [2019-09-10]. https://www.cs.umd.edu/class/fall2017/cmsc818O/papers/aeg.pdf

[39] Bisht P, Hinrichs T, Skrupsky N, et al. WAPTEC: Whitebox analysis of Web applications for parameter tampering exploit construction [C] //Proc of the 18th ACM Conf on Computer and Communications Security (CCS'11). New York: ACM, 2011: 575—586

[40] Cha S K, Avgerinos T, Rebert A, et al. Unleashing mayhem on binary code [C] //Proc of 2012 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2012: 380—394

[41] Gallingani D. Static detection and automatic exploitation of intent message vulnerabilities in Android applications [J/OL]. [2019-09-10]. https://www.politesi.polimi.it/bitstream/10589/92648/3/tesi.pdf

[42] Böttinger K, Eckert C. DeepFuzz: Triggering vulnerabilities deeply hidden in binaries [G] //LNCS 9721: Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2016: 25—34

[43] You Wei, Zong Peiyuan, Chen Kai, et al. SemFuzz: Semantics-based automatic generation of proof-of-concept exploits [C] //Proc of the 2017 ACM SIGSAC Conf on Computer and Communications Security (CCS'17). New York: ACM, 2017: 2139—2154

[44] Wu Wei, Chen Yueqi, Xu Jun, et al. FUZE: Towards facilitating exploit generation for kernel use-after-free vulnerabilities [C] //Proc of the 27th Security Symp (Security 18). Berkeley, CA: USENIX Association 2018: 781—797

[45] Wang Yan, Zhang Chao, Xiang Xiaobo, et al. Revery: From proof-of-concept to exploitable [C] //Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security (CCS'18). New York: ACM, 2018: 1914—1927

[46] Huang S K, Huang M H, Huang P Y, et al. Software crash analysis for automatic exploit generation on binary programs [J]. IEEE Transactions on Reliability, 2014, 63(1): 270—289

[47] Brooks T N. Survey of automated vulnerability detection and exploit generation techniques in cyber reasoning systems [C] //Proc of Intelligent Computing. Berlin: Springer, 2018: 1083—1102

[48] Padaryan V A, Kaushan V V, Fedotov A N. Automated exploit generation for stack buffer overflow vulnerabilities [J]. Programming and Computer Software, 2015, 41(6): 373—380

[49] Grieco G, Mounier L, Potet M L, et al. A stack model for symbolic buffer overflow exploitability analysis [C] //Proc of the 6th 2013 IEEE Int Conf on Software Testing, Verification and Validation Workshops. Piscataway, NJ: IEEE, 2013: 216—217

[50] Krupp J, Rossow C. Teether: Gnawing at ethereum to automatically exploit smart contracts [C] //Proc of the 27th Security Symp. Berkeley, CA: USENIX Association, 2018: 1317—1333

[51] Xu Luhang, Jia Weixi, Dong Wei, et al. Automatic exploit generation for buffer overflow vulnerabilities [C] //Proc of 2018 IEEE Int Conf on Software Quality, Reliability and Security Companion (QRS-C). Piscataway, NJ: IEEE, 2018: 463—468

[52] Caselden D, Bazhanyuk A, Payer M, et al. Transformation-aware exploit generation using a HI-CFG [R]. Fort Belvoir, VA: Defense Technical Information Center, 2013

[53] Zhang Rui, Deutschbein C, Huang Peng, et al. End-to-end automated exploit generation for validating the security of processor designs [C] //Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2018: 815—827

[54] Ji Tiantian, Wu Yue, Wang Chang, et al. The coming era of alphahacking: A survey of automatic software vulnerability detection, exploitation and patching techniques [C] //Proc of the 3rd IEEE Int Conf on Data Science in Cyberspace. Piscataway, NJ: IEEE, 2018: 53-60

[55] Ruffell M. Applying bytecode level automatic exploit generation to embedded systems [J/OL]. [2019-09-10]. https://ir.canterbury.ac.nz/handle/10092/14451

[56] Ruffell M, Hong J B, Kim H, et al. Towards automated exploit generation for embedded systems [G] //LNCS 10144: Information Security Applications. Berlin: Springer, 2017: 161-173

[57] Alhuzali A, Gjomemo R, Eshete B, et al. NAVEX: Precise and scalable exploit generation for dynamic Web applications [C] //Proc of the 27th Security Symp. Berkeley, CA: USENIX Association, 2018: 377-392

[58] Kang J, Park J H. A secure-coding and vulnerability check system based on smart-fuzzing and exploit [J]. Neurocomputing, 2017, 256: 23-34

[59] Luo Lannan, Liu Peng, Zeng Qiang, et al. System service call-oriented symbolic execution of Android framework with applications to vulnerability discovery and exploit generation [C] //Proc of the 15th Annual Int Conf on Mobile Systems, Applications, and Services (MobiSys'17). New York: ACM, 2017: 225-238

[60] Wang Minghua, Su Purui, Li Qi, et al. Automatic polymorphic exploit generation for software vulnerabilities [G] //LNCS 127: Security and Privacy in Communication Networks. Berlin: Springer, 2013: 216-233

[61] Alhuzali A. Automatic Exploit Generation for Web Applications [D]. Chicago: University of Illinois, 2018

[62] Huang S K, Huang M H, Huang P Y, et al. CRAX: Software crash analysis for automatic exploit generation by modeling attacks as symbolic continuations [C] //Proc of the 6th 2012 IEEE Int Conf on Software Security and Reliability. Piscataway, NJ: IEEE, 2012: 78-87

[63] Repel D, Kinder J, Cavallaro L. Modular synthesis of heap exploits [C] //Proc of the 2017 Workshop on Programming Languages and Analysis for Security (PLAS'17). New York: ACM, 2017: 25-35

[64] Liu Danjun, Wang Jingyuan, Rong Zelin, et al. Pangr: A behavior-based automatic vulnerability detection and exploitation framework [C] //Proc of the 17th IEEE Int Conf on Trust, Security and Privacy In Computing and Communications/the 12th IEEE Int Conf on Big Data Science and Engineering (TrustCom/BigDataSE). Piscataway, NJ: IEEE, 2018: 705-712

[65] Alhuzali A, Eshete B, Gjomemo R, et al. Chainsaw: Chained automated workflow-based exploit generation [C] // Proc of the 2016 ACM SIGSAC Conf on Computer and Communications Security (CCS'16). New York: ACM, 2016: 641-652

[66] Jan S, Panichella A, Arcuri A, et al. Automatic generation of tests to exploit XML injection vulnerabilities in Web applications [J]. IEEE Transactions on Software Engineering, 2019, 45(4): 335-362

[67] Huang S K, Lu Hanlin, Leong W M, et al. CRAXweb: Automatic Web application testing and attack generation [C] //Proc of the 7th IEEE Int Conf on Software Security and Reliability. Piscataway, NJ: IEEE, 2013: 208-217

[68] Gallingani D, Gjomemo R, Venkatakrishnan V N, et al. Practical exploit generation for intent message vulnerabilities in Android [C] //Proc of the 5th ACM Conf on Data and Application Security and Privacy (CODASPY'15). New York: ACM, 2015: 155-157

[69] Garcia J, Hammad M, Ghorbani N, et al. Automatic generation of inter-component communication exploits for Android applications [C] //Proc of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017). New York: ACM, 2017: 661-671

[70] Wan Yunpeng, Deng Yi, Shi Donghui, et al. Automatic exploit generation system based on symbolic execution [J]. Computer Systems & Applications, 2017, 26(10): 44-52 (in Chinese)
(万云鹏, 邓艺, 石东辉, 等. 基于符号执行的自动利用生成系统[J]. 计算机系统应用, 2017, 26(10): 44-52)

[71] Zhou Mingsong, Zeng Fanping, Zhang Yu, et al. Automatic generation of capability Leaks' exploits for Android applications [C] //Proc of 2019 IEEE Int Conf on Software Testing, Verification and Validation Workshops (ICSTW). Piscataway, NJ: IEEE, 2019: 291-295

[72] Yang Guangliang, Huang J, Gu Guofei. Automated generation of event-oriented exploits in Android hybrid apps [C] //Proc of 2018 Network and Distributed System Security Symp. San Diego, CA: Internet Society, 2018

[73] Yun I, Kapil D, Kim T. Automatic techniques to systematically discover new heap exploitation primitives [J/OL]. [2019-06-10]. https://arxiv.org/abs/1903.00503

[74] Jajodia S, Noel S, O'berry B. Topological analysis of network attack vulnerability [G] //Managing Cyber Threats. Berlin: Springer, 2005: 247-266

Zhao Shangru, born in 1995. Master candidate. His main research interests include network and information system security.

Li Xuejun, born in 1969. PhD, associate professor, master supervisor. Her main research interests include attribute-based encryption, information security, Internet of things.

Fang Yue, born in 1997. Master candidate. His main research interests include network and information system security.

Chen Kai, born in 1982. PhD from the University of Chinese Academy of Science in 2010. Professor. His main research interests include software analysis and testing, machine learning and privacy.

Yu Yuanping, born in 1994. PhD candidate. Her main research Interests include software and system security.

Su Purui, born in 1976. PhD, professor. His main research interests include software and system security, malware analysis and vulnerability discovery.

Huang Weihao, born in 1993. PhD candidate. His main research interests include automatic exploit generation(AEG), program analysis, trusted computing based on SGX and software security.

Zhang Yuqing, born in 1966. PhD, professor. His main research interests include network and information system security.