计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220410
61(9): 2261−2274, 2024

软件漏洞自动化利用综述

武泽慧[1] 魏 强[1] 王新蕾[1] 王允超[1] 燕宸毓[1] 陈 静[2]
[1](数学工程与先进计算国家重点实验室(战略支援部队信息工程大学) 郑州 450001)
[2](郑州大学网络空间安全学院 郑州 450001)
(wuzehui2010@foxmail.com)

Survey of Automatic Software Vulnerability Exploitation

Wu Zehui[1], Wei Qiang[1], Wang Xinlei[1], Wang Yunchao[1], Yan Chenyu[1], and Chen Jing[2]
[1](*State Key Laboratory of Mathematical Engineering and Advanced Computing* (*Strategic Support Force Information Engineering University*), *Zhengzhou* 450001)
[2](*School of Cyber Science and Engineering*, *Zhengzhou University*, *Zhengzhou* 450001)

Abstract In recent years, the number of software vulnerabilities has increased sharply and its harmfulness has aroused widespread concern in society. Compiling vulnerability utilization code accurately, efficiently and quickly is the key to vulnerability damage assessment and vulnerability repairment. At present, the vulnerability exploitation code mainly relies on manual analysis and writing, which is inefficient. Therefore, how to realize automatic vulnerability exploitation code generation (AEG) is a hotspot and difficulty in software security research field. In this paper, we analyze the representative achievements in this field in recent 30 years. Firstly, we divide the vulnerability automatic utilization process into four typical segments: vulnerability root location, reachable path search, vulnerability primitive generation and utilization code generation. After that we sort out and select the typical work of the above achievements from the three perspectives of human-machine boundary, attack and defense game, and common basic technology. And on this basis, we define the key points, difficulties and phased achievements of the current research. Finally, from the gap between the existing achievements and the practical application of automatic exploit generation, we discuss the bottleneck problems existing in the current research, the future development trend of AEG, and the next research points we should focus on.

Key words software security; vulnerability analysis; automatic exploit; exploit generation; vulnerability root cause

摘 要 近年来软件漏洞数目急剧增加,漏洞危害也引起业界广泛关注.准确、高效、快速地编写出漏洞利用代码是漏洞危害性评估和漏洞修复的关键.当前漏洞利用代码主要依赖人工手动分析编写,效率较低.因此,如何实现自动化的漏洞利用代码生成是该领域研究的热点和难点.综述分析了该领域近30年的代表性成果,首先将漏洞自动化利用过程分为典型的4个环节:漏洞根源定位、可达路径搜索、漏洞原语生成、利用代码生成.然后从人机边界、攻防博弈、共性技术3个角度对上述成果进行梳理,明确当前研究的重点、难点,以及取得的阶段性成果.最后从现有成果与技术实用化所面临的差距方面,论述当前研究存在的瓶颈问题、未来的发展趋势,以及下一步的研究重点.

关键词 软件安全;漏洞分析;自动化利用;利用生成;漏洞根源

中图法分类号 TP393

收稿日期:2022-05-21;**修回日期**:2023-12-19

基金项目:国家重点研发计划项目(2019QY0501)

This work was supported by the National Key Research and Development Program of China (2019QY0501).

随着信息技术的发展, 作为信息交互的主要载体, 软件已经成为日常生活中不可缺少的一部分, 其可靠性、安全性也愈发得到人们的关注. 然而, 随着软件功能的逐渐丰富, 其代码结构愈发复杂, 代码规模和种类日益增大, 软件内部存在的漏洞数量也急剧增多, 根据 2021 年国家信息安全漏洞共享平台 CNVD(China national vulnerability database)统计, 软件应用程序漏洞占到了总量的 46.6%, 位列影响对象分类统计的第一.

软件漏洞通常表现为一段可被攻击者利用的代码, 漏洞本身不会造成损害. 漏洞产生损害往往是因为漏洞被攻击者利用, 编写出可以造成损害的漏洞利用代码, 攻击者使用该代码实施恶意行为. 如 2017 年 WannaCry 勒索病毒利用“永恒之蓝”漏洞洗劫全球 150 多个国家, 造成严重破坏; 2019 年 WinRAR 路径穿越漏洞构造恶意压缩文件达到代码执行效果, 影响全球超过 5 亿用户; 2021 年 Windows Print Spooler 服务之中存在 PrintNightmare 漏洞, 攻击者利用此漏洞在目标设备上远程执行任意代码, 所有版本的 Windows 系统均受影响. 上述这些事件严重危害了国家网络安全、关键基础设施安全及重要行业的信息安全, 已经或者将要造成重大损失.

因此, 自漏洞出现以来, 漏洞利用技术的研究即成为该领域研究的焦点, 而漏洞自动化利用则是众多研究方向中最具挑战性的一个. 漏洞自动化利用是指由计算机自动生成一段代码, 该代码可以在受害者不知道的情况下触发漏洞, 完成攻击者预期行为. 本文将漏洞自动化利用过程分为 4 个步骤: 1)分析漏洞成因, 定位漏洞根源; 2)确定利用点, 搜索构造利用路径; 3)消除副作用, 编写漏洞利用原语(通常指具备攻击能力的利用单元或原子操作, 表现形式为漏洞程序的一个特定输入, 为具体攻击代码提供组成单元); 4)突破防护机制, 编写漏洞利用代码, 漏洞自动化利用过程如图 1 所示.

当前漏洞利用普遍依赖手工分析, 自动化利用技术需要克服的难题较多, 目前适用的场景比较单一(主要集中在 CTF 类的比赛中, 漏洞类型多以栈溢出、格式化字符串这种成因简单的类型为主), 以及软件对象的类型和规模有限(多以 4 MB 以下文件处理类程序为主). 因此在实际应用方面, 主要用于辅助漏洞危害性评估, 也可用于辅助漏洞修复. 但是类似于雕版印刷必然取代手工临摹一样, 漏洞自动化利用过程可以分解为 4 个标准化步骤, 如图 1 所示, 每个步骤内以及步骤之间的操作, 均可通过技术实现自动化. 因此, 漏洞自动化利用也必将取代当前以手工分析为主的漏洞利用方法. 因此, 如何实现利用过程的自动化, 是当前学术界研究的热点, 研究的漏洞类型从最简单的栈溢出到复杂的 UAF(use-after-free), 软件对象从简单的 CTF(capture the flag)赛题到复杂的 Linux 内核, 安防机制也实现了从 Canary 绕过到 DEP(data execution protection)和 ASLR(address space layout randomization)对抗的升级[1-2]. 本文分析了

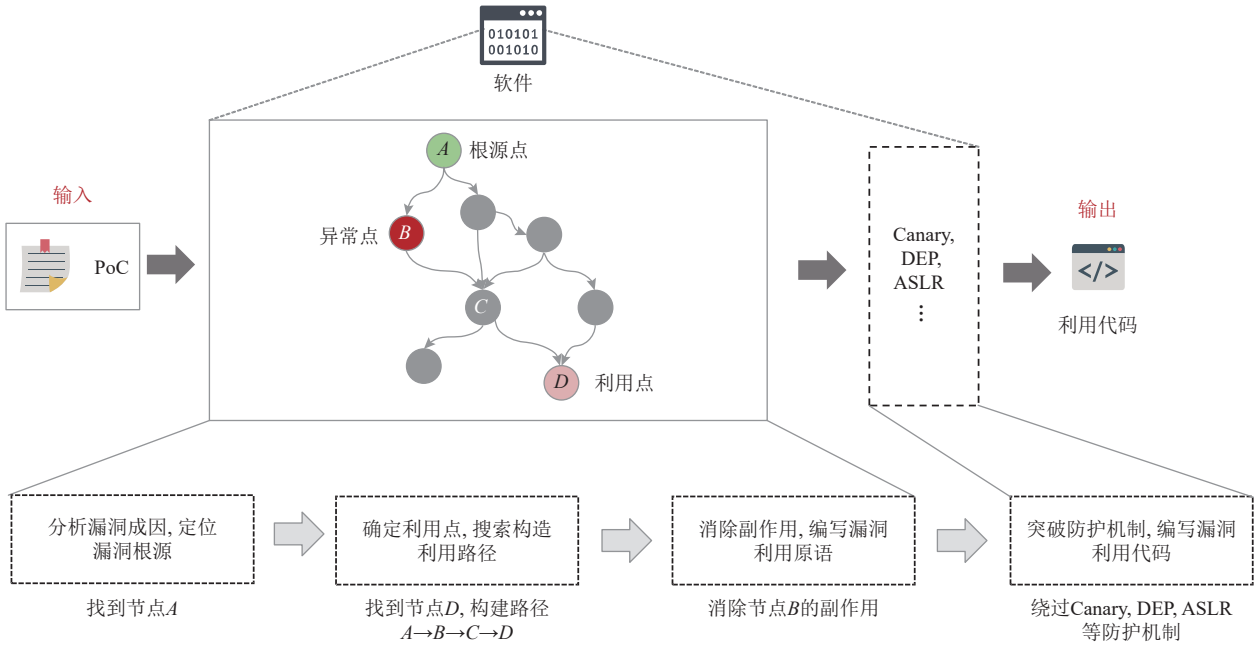

Fig. 1 Flowchart of automatic vulnerability exploitation

图 1 漏洞自动化利用流程图

该领域近30年的研究成果，重点梳理近10年在IEEE, ACM, Springer等期刊和安全会议中的代表性文献，从人机边界、攻防博弈、共性技术3个方面对成果进行归类和比较分析，尝试总结当前研究成果存在的不足，指出该领域可能的研究方向.

1 人机边界

纵观漏洞利用技术的发展历史，凡是漏洞利用技术遇到了瓶颈问题、新漏洞类型出现亟需新的利用思路、新的防护机制亟待绕防与突破时，都依赖“天才般”的灵感去创造新的利用技术和技巧，而人在其中起到了“化腐朽为神奇”的作用，甚至是采用“切换视角”的方式用超越该问题维度的思维去解决该问题. 虽然“人”在漏洞利用中起着最为关键的作用，但是在涉及到计算、枚举、过程存储等环节时，“计算机”也起到了重要的辅助作用. 重复性、高算力的操作由“机器”完成，决策性、高智力参与的操作由“人”完成，“智力的参与度”决定了“人”与“机器”的协作“边界”[3]. 因此，从“人”与“计算机”在漏洞利用过程中所扮演角色的角度，可以将漏洞利用的发展过程分为2个阶段：依赖专家经验的人工漏洞利用阶段和基于模板知识的自动化漏洞利用阶段.

1.1 依赖专家经验的人工漏洞利用阶段

该阶段主要依赖于专家经验. 作为一个精巧的工作过程，漏洞利用代码的完成涉及到众多细节，由人工来完成可以最大程度保证漏洞利用的高可靠性. 因此，目前几乎具备实用化水平的漏洞利用工具都是由人工来完成的.

1988年，Morris利用UNIX Fingerd服务程序栈溢出漏洞实现了蠕虫病毒的快速传播，但是此时仅有少数研究人员理解内存布局，利用该类漏洞进行网络攻击的门槛较高. 1996年，Aleph[4]发表了“Smashing the Stack for Fun and Profit”，文中作者详细阐述了栈空间的结构以及溢出漏洞的利用技巧，降低了漏洞利用的门槛. 1997年，Smith在前人研究的基础上，收集了多种处理器架构下的Shellcode，并给出了在不同UNIX类系统中，利用栈溢出漏洞编写Exploit(漏洞利用代码)的详细过程. 1999年Dark等人将该方法成功移植到Windows系统中，自此漏洞利用成为一种多数攻击者可以掌握的网络攻击技术，利用漏洞也逐渐成为网络攻击的重要手段，如2000年和2001年微软IIS服务程序先后被爆出多个栈溢出漏洞，2003和2004年Windows RPC的2个漏洞更直接导致了“冲击波病毒”和“震荡波病毒”的快速传播.

2004年开始，为了对抗该类漏洞利用代码，UNIX类操作系统和Windows操作系统先后引入了多种内存保护机制，如GS, SafeSEH, DEP等. 特别是DEP机制的广泛使用，直接阻断了传统缓冲区溢出漏洞利用代码的执行，导致90%以上的Exploit失效. 但是攻击者们并未停止对新攻击手段的探索. 2005年，研究人员提出Borrow Code Chunks(代码块借用)的攻击思路，利用内存空间中已经存在的可执行代码拼装成攻击代码，可以有效绕过DEP机制的防护，2007年该类技术趋于成熟，围绕ROP(return oriented programming)技术实现DEP对抗的攻击方法先后被实际应用. 2010年，研究人员证明ROP是图灵完备的，自此ROP成为对抗DEP机制的一种通用型技术.

在攻防双方围绕DEP机制进行博弈的同时，2007年微软在Vista操作系统中正式采用ASLR防护机制，将系统防御能力提升到一个新的高度.ASLR增大了内存布局的随机性，而漏洞利用代码需要确定性的地址才能实现控制流劫持后的稳定跳转. 该技术的出现，再一次提高了漏洞利用的门槛. 尽管研究人员先后提出了多种稳定跳板地址信息泄露的方法，以及堆喷射、堆风水等ASLR对抗方法，但是至今未见对抗ASLR机制的通用型技术出现，ASLR等防护机制仍然是漏洞利用的关键屏障.

综上分析可知，该阶段围绕漏洞利用展开的攻防博弈中，新型漏洞的发现、防护机制的出现与突破、创新型利用思想的提出，均是来源于由人的灵感、智慧凝结的专家经验，如Dark提出了“Jmp esp”的技巧，让Windows操作系统下的缓冲区溢出漏洞利用打开了一片天空，缓冲区溢出对抗技术中，各种漏洞利用缓解技术和反利用缓解技术的突破，至今都是由人来突破的. ROP方法的提出将漏洞利用技术提升到一个新的高度，其后出现的一些自动化的方法，形成的一些基于文档和经验的共同利用原理均是以人工构造的模板为基础.

1.2 依赖机器的自动化漏洞利用阶段

在专家经验驱动新漏洞利用方法产生的同时，10多年来研究人员也在不断尝试基于已有知识沉淀下的漏洞自动化利用技术，研究如何在有限条件下实现自动化的漏洞利用代码生成. 研究人员认为：具有相同机理的漏洞，其利用过程具有高度相似性，只要能将这种共性提取总结出来，形成指导性的原则，就可以用于指导机器去构造完成相应的漏洞利用

代码. 其发展历经了 3 个阶段:

第 1 个阶段以 APEG[5] 为起点漏洞利用自动化的初步尝试. 2008 年基于补丁比对的漏洞利用代码自动生成方法 APEG 出现, 依赖“机器判断”的结果, 并由“机器生成”漏洞利用代码. 2009 年提出的 AXGEN[6]、2012 年提出的 Mayhem[7]、2013 年提出的 PolyAEG[8]、2014 年提出的 AEG[9]、2015 年提出的 FlowStitch[10] 等, 则是将“机器判断”提升至“机器分析”, 实现了“机器能力”的增强. 但是该类方法针对简单内存破坏类漏洞进行, 并且未考虑系统防护机制, 实用化程度低.

第 2 个阶段是以 CGC 为代表的实际应用. 2016 年, 由 DARPA 主办的 CGC 挑战赛吸引了全球安全研究人员的目光, 在人工编写的赛题上, 尝试目标探测、漏洞挖掘、漏洞分析、漏洞利用及漏洞修复等全流程自动化的可行性. 在该比赛中出现了比肩专家经验的漏洞利用“机器人”, 如升级版的 Mayhem, Mechanical Phish 等. CGC 比赛之后, 漏洞利用代码自动化生成的研究出现了新的热潮, 围绕不同软件对象, 采用不同技术组合, 针对多种系统防护机制的自动化利用技术先后出现. 该阶段的应用普遍以人工构造的漏洞案例为实验对象, 如 CGC 比赛中将系统调用数目进行限制(7 个系统调用), 仅能够在类似于 CTF 赛题的软件中实现自动化利用. 但是作为漏洞自动化利用的第 1 次实际应用, 也有力地推动了该领域的研究.

第 3 个阶段是后 CGC 时代. 该阶段中针对真实软件和系统防护机制(内核 SMEP, SMAP 等)自动化突破方面均有相关研究成果出现. 但是该阶段仍处于“延长线”性质的研究. 虽然覆盖的漏洞类型种类有所增加, 但对所处理问题的尺寸有限制, 如 FlowSwitch[10] , Q[11], CRAX[12], HCSIFTER[13], Revery[14], AutoExp[15] 等所能处理的程序规模基本都在 3 MB 以下. 此外, 漏洞利用不是简单的已有知识复用, 仅仅通过“机器分析”无法对精密复杂漏洞进行利用, 如机器无法自动化构造 CPU 熔断和幽灵漏洞的侧信道信息泄露状态以及内存探测代码, 更无法创造新的利用模板.

2 攻防博弈

自 1988 年第 1 次漏洞利用攻击出现以来, 先后有多种系统防御机制被提出, 同时攻击者也不断创造出新的漏洞利用方法, 系统防御与漏洞利用呈螺旋式对抗趋势. 图 2 展示了漏洞防护技术发展阶段图, 以操作系统为例, 2000 年以前, 对漏洞利用基本处于无防护阶段, 简单的缓冲区漏洞即可造成大范围的网络攻击. 2000—2010 年, GS, SafeSEHR, DEP, ASLR, Safe Unlink 等系统防御机制先后被提出, 每一次防御机制的出现, 都带来漏洞利用方法的升级. 这个阶段是攻击能力强于防御能力的阶段, 可被称为“弱防护阶段”. 2010 年以来, 随着硬件性能的革命性提升, 对用户使用体验影响较大的防御机制被操作系统默认开启, 并且更加严格的内核、权限、程序控制流等保护机制也先后适配到不同的操作系统中. 漏洞利用的难度越来越大, 这个阶段大有防御能力强于攻击能力的趋势, 可被称为“强防护阶段”.

系统防御机制的出现, 不仅给漏洞利用带来了革命性的变化, 也一直是漏洞自动化利用需要克服的难题. 针对不同的系统防御机制, 研究人员先后提出了多种绕过和对抗方法.

2014 年 Avgerinos 等人[9] 提出的基于源码的漏洞自动化利用方法(automatic exploit generation, AEG), 可以生成绕过 DEP 机制的漏洞利用代码, 通过 return-to-libc 的方式实现控制流劫持, 但是该方法无法突破 ASLR 的防护.

因此, 为解决 ASLR 机制给控制流劫持带来的障碍, Schwartz 等人[11] 于 2011 年的 USENIX Security 会议上, 提出了一种返回导向编程的代码自动化生成方法 Q. Q 的工作流程是先收集目标程序中的 Gadget,

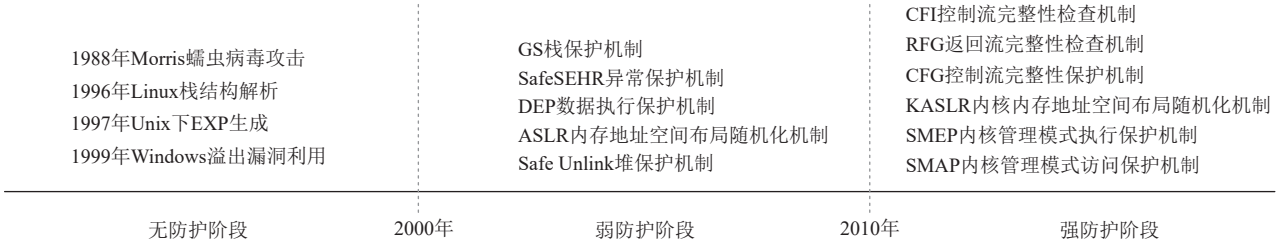

Fig. 2 Development stage diagram of vulnerability protection technology

图 2 漏洞防护技术发展阶段图

随后利用面向 Gadget 的编程方法，构建 ROP 漏洞利用链. 对 9 个小型真实软件漏洞进行实验，在开启 DEP 和 ASLR 机制后，Q 自动生成的漏洞利用代码仍然可以稳定执行.

上述方法是从控制流对抗的角度实现系统防御下的漏洞自动化利用. 2015 年 Hu 等人[10]提出的 FlowSwitch 则是从数据流的角度实现漏洞自动化利用. FlowSwitch 在维持原始程序控制流的基础下，对被测二进制程序中已发现的内存漏洞展开分析，并对正常数据流中敏感变量的内容做修改，通过该方法得出的漏洞利用代码被证实可达到提权和信息泄露等目的. 在实验中选取了 8 个带有漏洞的真实程序，结果表明 FlowSwitch 生成的 19 个漏洞利用代码均绕过了 DEP 以及细粒度 CFI 等系统保护机制.

FlowSwitch 和 Q 可针对用户态空间的系统防御机制进行自动化漏洞利用代码的生成. 2018 年，Wu 等人[16]提出针对内核 SMEP 防护机制下的漏洞自动化利用方法 FUZE. 该工具需要关闭 KASLR 后才能够实现稳定绕过. 在 FUZE 的基础上，2020 年 Chen 等人[17]提出 KASLR 绕过方法并开发了原型工具 ELOISE，该方法采用对象识别的方法筛选候选内存分配点，进而利用污点分析和约束求解生成漏洞利用代码.

2021 年 Liguori 等人[18]针对人工智能模型训练的问题，构建了 Linux 平台下的漏洞数据集 IA32_shellcode，该数据集主要通过真实漏洞平台 exploit-db 和 shell-storm，收集包含 3 200 个具有规范格式汇编指令的例子和对应的自然语言注释. 该数据集可用于自动生成 shellcode 的机器翻译任务，能够自动从自然语言翻译生成漏洞利用的 shellcode. 作为一次有效的尝试，该成果进一步推动了该方向的研究. 但目前该数据集的规模、规范性、多样性方面距离大规模模型训练的需要仍有一定差距.

2022 年 Yang 等人[19]提出了自动化 shellcode 生成模型 DualSC. 该文作者提出对偶学习的概念，将 shellcode 的自动生成形式化为对偶任务求解，使用浅层 Transformer 和对偶学习进行训练，并设计归一化方法调整以适应这些低资源任务（即训练数据不足），提高模型的泛化能力. 最终 DualSC 在 2 个任务中均强于目前最优秀的自动化 shellcode 生成模型，但是 DualSC 在真实漏洞利用中的可靠性还亟待验证. 近年来更加严格的系统防护机制先后被提出，如控制流完整性（control flow integrity, CFI）保护机制、返回流保护（return flow guard, RFG）机制、内核沙箱机制等[20-21]. 目前未见针对这些新型防护机制的突破方法，给漏洞自动化利用带来了更大的挑战.

3 共性技术

软件漏洞利用代码的自动构造需要基于程序分析. 随着漏洞挖掘领域成功应用补丁比对、污点分析、符号执行和模糊测试等技术的发展，研究者开始尝试将这些技术用于漏洞自动化利用. 由于在漏洞自动化利用方面，这些技术具有通用性、基础性的特点，本文将其归纳为“共性”技术. 图 3 展示了 2008—2021 年的典型漏洞自动利用研究成果，图中虚线左侧的研究围绕如何在“已劫持/单步可控制劫持流”的前提下实现自动化的漏洞利用代码生成，虚线右侧的研究则围绕如何基于“代码生成与原语发现”方法进一步实现自动化的漏洞利用代码生成. 其共性技术类别可分为图 3 所示的 7 种.

3.1 依赖机器的自动化漏洞利用阶段

补丁比对技术是攻击者常用的一种方法，其主要目的在于发现那些已经修复但尚未公开披露的漏洞，即 1Day 漏洞. 通常，安全公告或补丁说明中不会提及漏洞产生的根本原因，这给攻击者利用漏洞带来了困难. 然而，基于二进制文件在打补丁前后的对比差异可以确定产生漏洞的位置，并结合其他漏洞分析技术，进一步确定漏洞的根本原因，最终生成能够利用该漏洞的攻击代码.

在 IEEE S&P2008 年会议上 Brumley 等人[5]提出了一种基于二进制补丁比对的漏洞利用自动生成方法 APEG. 该方法的基本思想是，在补丁程序中引入了用于导致原始程序发生异常的过滤条件. 因此，APEG 仅需要定位补丁程序中添加的过滤条件并构造不满足这些条件的“违规”输入，这个输入就可以被视为原始程序的一个实现漏洞利用的可行输入.

APEG 尝试自动化构建漏洞利用的工作，不但该方法思路相对直接明了，而且由于其高度可操作性，因此广受研究人员的认可. 然而，APEG 方法存在 2 个主要不足：1）无法应对在补丁程序中未添加过滤判断的情况；2）支持能够自动实现利用的漏洞类型主要为拒绝服务类漏洞利用，这种利用仅导致原程序崩溃，无法实现控制流劫持的目标.

3.2 基于数据流分析的漏洞自动化利用

数据流分析是一种信息分析技术，其目的在于获取程序执行路径上相关数据的信息. 由于程序数据流的数据依赖和控制依赖关系可能与漏洞紧密相

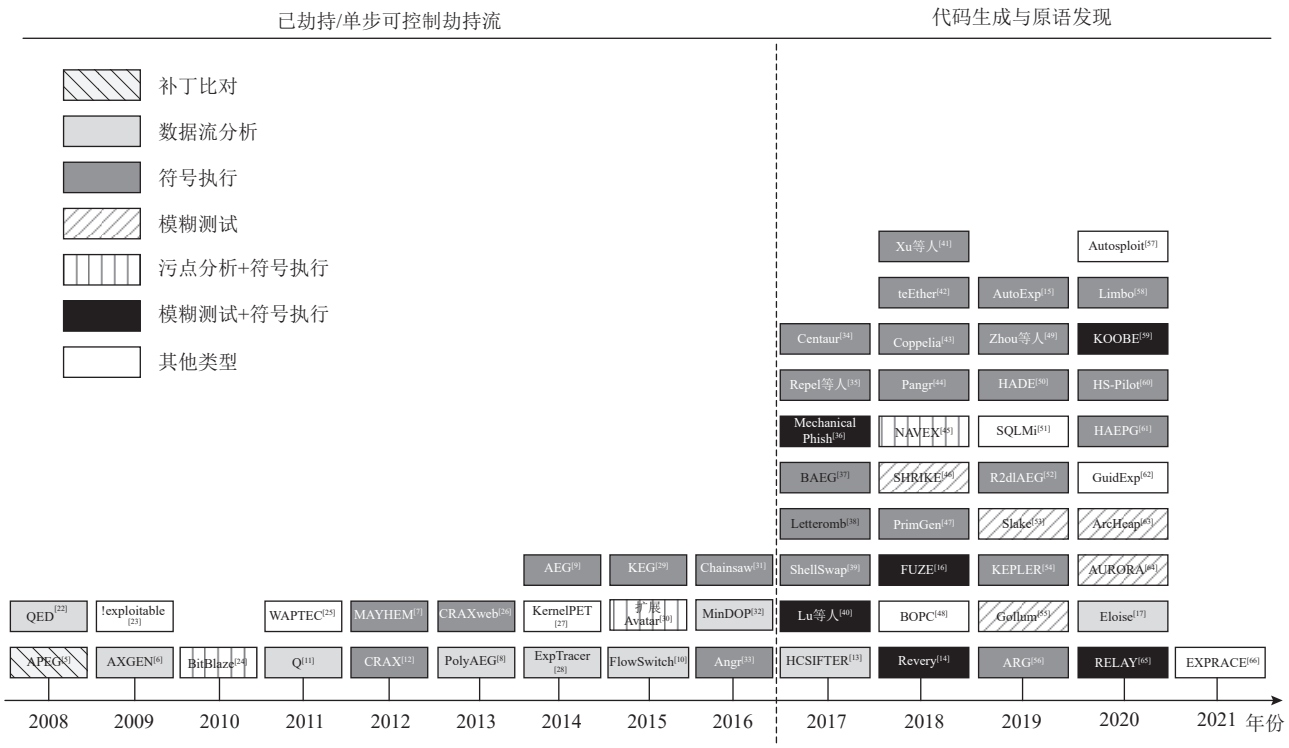

Fig. 3 Classification diagram of typical vulnerability automatic utilization research results

图 3 典型漏洞自动化利用研究成果分类图

关，因此也被用于漏洞分析. 数据流分析可对多种漏洞和缺陷进行检测分析，同时能够对其他漏洞分析方法提供支撑. 典型的基于数据流分析的自动漏洞利用工具包括 AXGEN[6]，PolyAEG[8]，Q[11]，HCSIFTER[13]，Eloise[17]，QED[22]，ExpTracer[28]，MinDOP[32] 等. 将数据流分析应用到漏洞自动化利用中，需要涉及对程序进行数据结构恢复. 该过程需要消除程序对寄存器、栈帧等底层环境的依赖，对函数定义、引用、参数和返回值等进行恢复. 然而，在进行二进制程序的跨平台恢复时，存在无法完全还原用户自定义过程中产生的参数、变量等数据的问题. 为解决这一问题，中国科学院的马湘宁等人[67] 于 2002 年提出了一种静态二进制翻译系统（XM 翻译器）上的过程恢复技术. 该技术主要分为参数恢复及返回值恢复 2 个核心. 此外，他们还提出了一种新的简便方法来解决返回值恢复的问题，避免了传统方法中需要开展复杂的数据流和控制流分析. 该方法基于 x86 指令集本身的调用特性和实例进行分析，为实现返回值恢复提供了一种简单的方案. 2006 年，解放军信息工程大学的付文等人[68] 针对 IA64 体系结构的特点及其对过程调用的相关规定进行了研究，实现了对 IA64 过程的恢复. 同时，在大量的实践基础上，提出了一种浮点参数恢复的优化方法. 2009 年，解放军信息工程大学的方霞等人[69] 发现 IA32 体系结构中可执行程序在反编译过程存在寄存器参数恢复问题，并开展了针对性研究，他们基于 IA32 体系下的二进制函数调用约定，分析了寄存器参数的特征，并借鉴了数据流分析中的到达-定义分析和使用-定义链等经典方法. 通过利用寄存器定义和使用等信息，提出了一种识别、处理和实参恢复的寄存器参数方法.

2008 年，斯坦福大学的 Martin 等人[22] 提出针对 Web 应用程序的 XSS 和 SQL 注入漏洞的自动漏洞利用工具 QED，采用污点分析技术标记不受信任的数据来识别漏洞，并根据污点数据的路径构造漏洞利用.

2009 年，Heelan[6] 设计了基于异常输入的自动化漏洞利用方法 AXGEN，该方法结合了数据流分析和动态污点分析等技术，专注于针对缓冲区溢出类漏洞的利用程序的生成.

Schwartz 等人[11] 在 2011 年的 USENIX 会议上，设计了一种高可靠性漏洞利用的 ROP 代码自动生成方法 Q. Q 的核心工作逻辑是在被测程序中提取 gadget，并使用面向 gadget 的语言自动化编程生成 ROP. 类似的方法还包括 MinDOP[32].

2013 年，Wang 等人[8] 设计了一个自动化漏洞利用工具 PolyAEG.该工具基于污点分析技术来进行动

态分析并获取被测二进制程序中所有潜在的控制流劫持点. 研究人员通过构造多种跳转指令链以及对收集的路径约束进行求解, 从而生成了多样化的漏洞利用样本. 类似的方案有 HCSIFTER[13], ELOISE[17], ExpTracer[28].

2015 年, 新加坡国立大学的 Hu 等人[10] 设计了一种自动化漏洞利用方法——FlowSwitch, 该方法基于数据流分析. 其核心原理是在维持程序原始控制流的情况下, 基于对被测二进制程序中已发现的内存漏洞开展分析, 通过修改原始数据流中关键变量的内容, 以生成有效的漏洞利用代码, 从而完成对目标主机的信息泄露和提权等功能.

3.3 基于符号执行的漏洞自动利用

符号执行是一类程序分析技术, 它基于对程序的分析来获取触发指定区域代码执行的输入. 在符号执行中, 目标程序的输入视为符号变量. 当程序执行时, 数据被视为条件表达式或其他操作的等效物, 它的结果被逐步表示为对输入值的约束, 以确保代码按照指定路径执行. 每当代码执行存在符号值的条件检查时, 将需要进行分支执行, 在真实路径上增加已满足的约束, 而在其他路径上添加未满足的约束. 通过使用约束求解器最终寻找满足约束的具体值, 从而生成程序的测试用例. 符号执行被广泛应用于自动化领域.

自 2008 年提出起, 直至最新的研究, 符号执行在漏洞自动化利用中扮演重要角色. 常见的基于符号执行的自动漏洞利用工具有 MAYHEM[8]、AEG[9]、CRAX[12]、Craxweb[26]、Chainsaw[31] 、Angr[33]、Repel 等人[35]、Letteromb[38]、teEther[42]、Pangr[44]、PrimGen[47]、KEPLER[54] 等.

2012 年的 IEEE S&P 会议上, Avgerinos 及其团队[7] 提出了一种名为 AEG 的有效漏洞自动挖掘和利用方法. 该方法使用了程序验证技术, 寻找能够导致程序进入非安全状态且可被利用的输入. 这些非安全状态包括内存越界写入和恶意格式化字符串等情况; 而可被利用主要指的是程序的 EIP(执行指令指针) 可被任意篡改. AEG 方法结合了经过优化的动态指令插桩和符号执行技术, 实现了软件漏洞从自动发现到自动利用的完整过程. 此外, AEG 输出的利用样本直接具备劫持控制流的能力, 是一个真正面向控制流漏洞利用的自动化构建方法. 相似的论文有 CRAX[12]、AutoExp[15]、BAEG[37]、Shellswap[39]、Pangr[44]、Xu 等人[41] 所提方法、R2dlAEG[52] 等.

2012 年, Cha 等人[7] 设计了漏洞利用自动生成方法 Mayhem, 该方法面向二进制程序.Mayhem 结合了在线式符号执行中的速度快和离线式符号执行中的内存负载低 2 个特点并提出一种新的内存模型, 该模型基于索引使得这种漏洞利用自动化方法更加实用.

2013 年, Huang 等人[26] 针对 Web 应用程序提出了自动漏洞利用工具 Craxweb, 通过检测对 SQL 服务器的符号查询或对 HTTP 服务器的符号响应来测试具有初始随机输入的 Web 应用程序. 在检测到符号输出之后, 利用 S2E 符号执行生成攻击字符串并再现结果, 从而模拟手动攻击行为. 类似的技术有 Chainsaw[31], PrimGen[47].

2015 年, Do 等人[29] 提出了 KEG 方法, 该方法能自动生成面向对象程序中信息流泄露漏洞利用代码. KEG 将符号执行与自合成整合起来, 为选定的程序位置的安全级别规范和信息流策略组成一个不安全公式. 之后通过不安全公式产生一个模型, 用以生成漏洞利用的输入数据. 类似的技术有 HAEPG[61].

2016 年, 加州大学的 Shoshitaishvili 等人[33] 在其研究中引入了面向二进制软件攻防的基础分析框架平台 Angr, 此平台通过模块化方法将现有的基础性软件分析方法实施起来, 并构建了一个功能齐全的分析框架. 这个框架涵盖了动态分析执行、控制流图生成、符号执行以及值域分析等诸多内容.

2017 年, 伦敦大学的 Repel 等人[35] 针对堆漏洞的利用问题, 提出了自动利用方法, 在堆管理器或目标应用程序中, 定位内存指针的控制流转移, 并对符号输入施加约束, 以便利用原语劫持指针. 最后合成漏洞利用的 payload, 并发出驱动代码, 将其反馈给应用程序. 类似的技术有 HADE[50], HS-Pilot[60].

2017 年, Garcia 等人[38] 在 FSE 会议上针对 Android 应用程序漏洞开发了自动漏洞利用工具 Letteromb. Letteromb 从漏洞识别点识别的易受攻击的语句开始执行反向静态符号执行 (SSE), 以确定 Intent 的负载, 当发送到易受攻击的组件时, 可能会执行每个易受攻击的语句. 类似的技术有 Zhou 等人[49] 所提技术、Centaur[34] 等.

2018 年, Zhang 等人[43] 提出一种面向 RTL 硬件程序漏洞的自动化利用代码生成方法 Coppelia.该方法首先使用 Verilator 将 RTL Verilog 代码翻译成语义相同的 C++代码. 然后, 在测试平台中设置安全关键断言, 并使用 Clang 编译器将新翻译的设计编译成 LLVM 字节码. 最后, 该方法定义了违背安全关键断言的处理器状态, 并利用符号执行技术定位到能够

使目标软件从初始状态到达违反状态的漏洞路径.

2018 年 USENIX 顶会上 Krupp 等人[42]针对智能合约的研究提出了名为 teEther 的自动漏洞利用工具. 该工具首先将以太坊虚拟机(EVM)的字节码进行分解, 并据此重新构建了相应的控制流图. 接着, 对该控制流图进行分析, 以获取关键指令和状态改变指令的信息. 随后, 通过探索路径和运用符号执行的方法, 生成了针对路径的约束条件. 最后, 借助生成模块解决了组合约束, 从而生成了对应的漏洞利用代码.

2019 年北京理工大学的 Wei 等人[56]针对面向返回编程 ROP 提出了自动 ROP 链生成工具 ARG. ARG 基于有向无环图 (directed acyclic graph, DAG) 的特性, 分析输入(可用 Gadget 集)的语义, 使用符号执行技术自动生成 ROP 链.

2019 年, 宾夕法尼亚州立大学的 Wu 等人[54]针对广泛应用于 Linux 内核中的利用缓解机制展开了研究. 他们提出了一种名为“single-shot”的 Linux 内核利用技术, 并基于部分符号执行技术, 开发了名为 KEPLER 的原型系统, 用于绕过缓解机制.

2020 年卡内基梅隆大学 Schwartz 等人[58]提出了一种通用的框架 Limbo 来在可执行文件中自动地实现代码复用攻击.Limbo 查找代码重用攻击的问题可简化为 SMC 问题, 使用可执行级别的 concolic 执行来解决该问题.

2021 年 Wang 等人[70]提出了一种绕过漏洞利用缓解措施生成可用漏洞代码的自动化解决方案 AEMB. 首先根据程序执行环境识别当前系统中的漏洞利用缓解措施, 然后通过符号执行来收集符号信息, 最后使用求解器求解得到漏洞利用代码. 证明了 AEMB 可自动绕过漏洞利用缓解措施, 并为 13 个应用程序中的 11 个成功生成实际可用的漏洞利用脚本. 但受限于约束求解器的求解能力, AEMB 在大型复杂程序中求解约束会出现路径爆炸的问题, 无法得到可用的漏洞利用代码.

3.4 基于模糊测试的漏洞自动利用

模糊测试(Fuzzing)系一种常见的软件测试技术, 它的核心思想是首先自动生成或半自动生成测试样例, 然后将这些样例输入到程序中, 观察程序是否存在异常的行为, 以发现潜在的程序错误. 安全研究人员常使用模糊测试来挖掘计算机软件或系统的安全漏洞. 经典的模糊测试流程包括 3 个环节: 测试样例生成、崩溃状态检测和测试样例裁剪. 例如, FuzzGen[71], SIoTFuzzer[72] 等都是以此测试流程为基础的代表性成果. 首先, 模糊测试通过变异等方法生成符合输入规范的样例, 并检查测试样例中所有可修改的部分. 在执行过程中, 评估测试用例的执行是否会产生可被利用的影响. 一旦探测到新的漏洞时, 将其输入样本进行最小化等价替换, 并生成一个包含基本操作组的 PoC 代码, 用作证明. 基于模糊测试的自动漏洞利用的代表性工具有 SHRIKE[46], Slake[53], Gollum[55], ArcHeap[63], AURORA[64] 等.

2018 年, 哈佛大学 Heelan 等人[46]针对堆内存损坏漏洞提出基于伪随机黑盒搜索的自动漏洞利用工具 SHRIKE, 利用模糊测试从 PHP 的回归测试开始, 发现与解释器堆交互的 PHP 代码片段, 然后使用搜索算法将这些片段拼凑成程序, 搜索实现所需堆布局的程序.

2019 年, Heelan[55]在 CCS 会议上提出了针对解释器进行自动堆漏洞利用的工具 Gollum, 挖掘代码中可能存在漏洞原语片段, 使用懒惰解析算法和遗传算法进行堆布局, 最终将算法集成到一个灰盒模块化的自动化漏洞利用工具.

2019 年, 宾夕法尼亚州立大学的 Chen 等人[53]针对内核系统调用和缺乏 SLAB 布局知识, 通过扩展 LLVM 和 Syzkaller 实现了 Slake, 主要分为 2 步: 构建内核对象数据库和调整 SLAB 内存布局. Slake 不仅可以使内核漏洞的利用方式多样化, 而且有时还会提升内核漏洞的可利用性.

2020 年, 波鸿鲁尔大学的 Blazytko 等人[64]针对二进制可执行文件崩溃的根源进行分析并实现了工具 AURORA, 首先创建一个多样化的程序行为的数据集, 然后监控相关的输入行为, 使用数据推断预测程序是否会崩溃.

2020 年, 佐治亚理工学院的 Yun 等人[63]提出了一个可以自动化地系统性挖掘堆利用原语的工具 ArcHeap, 其遵循经典模糊化中的一种常见范式: 测试生成、崩溃检测和测试缩减, 但它是为堆利用而定制的. 每当 ArcHeap 发现新的漏洞利用时, 它会最小化堆操作并生成 PoC 代码.

3.5 基于污点分析和符号执行相结合的漏洞自动利用

单独利用符号执行进行自动漏洞利用可能会面临路径爆炸、约束求解困难等问题, 因此一些论文开始关注将污点分析与符号执行相结合进行自动漏洞利用, 主要工具包括 BitBlaze[24]、扩展 Avatar[30]、NAVEX[45] 等.

2010 年, UCBerkely[24]开发了基于二进制信息的分析平台 BitBlaze, 该平台综合了二进制程序的信息流分析、污点分析和符号执行技术. 其主要目标是确

定崩溃时受攻击者输入文件控制的寄存器和内存地址，并通过对 NULL 指针进行切片以快速排除可利用性.

2016 年，坎特伯雷大学的 Ruffell 等人[30]针对嵌入式系统进行自动化漏洞利用研究，通过扩展现有的动态分析框架 Avatar，在目标设备上标记污点，利用污点分析执行直到硬件被初始化，然后将执行移动到仿真器以实施符号执行，实现了一种嵌入式系统固件漏洞利用代码自动生成的方法.

2018 年，伊利诺伊大学的 Alhuzali 等人[45]针对 Web 应用程序提出了自动漏洞利用工具 NAVEX，其采用符号执行创建 Web 应用程序各个模块的行为模型作为指导，并利用污点分析标记出包含潜在漏洞点的模块，结合动态分析自动验证漏洞并构造可用 Exploit.

3.6 基于模糊测试和符号执行相结合的漏洞自动利用

基于模糊测试与符号执行结合的自动漏洞利用代表性工具有 Revery[14]、FUZE[16]、Mechanical Phish[36]、Lu 等人[40]所提工具、RELAY[65]、KOOBE[59]等.

2016 年，加州大学圣芭芭拉分校的安全团队的 Shellphish 等人提出了一种名为 Mechanical Phish 的自动化漏洞利用方法[36]. 该方法使用了结合动态符号执行和 AFL 的漏洞挖掘引擎 Driller 来进行漏洞自动化挖掘. 随后借助基于 Angr 的漏洞利用生成引擎 Rex 来实现漏洞利用样本自动生成. 最后，基于补丁生成引擎 Patcher-ex 得到补丁程序.

2017 年，佐治亚大学的 Lu 等人[40]针对内核栈变量未初始化漏洞利用问题，提出自动化定向栈喷射方法，采用定制的符号执行和引导型模糊测试来识别内核输入和利用 Linux 内核栈变量未初始化漏洞. 类似的技术有 RELAY[65].

2018 年，Wu 等人[16]提出了基于内核 UAF 漏洞的自动化方法 FUZE. 该方法将符号执行、模糊测试、动态追踪等技术结合在一起，针对内核 UAF 漏洞自动生成漏洞利用样本. 类似的技术有 KOOBE[59].

为了解决程序崩溃现场无法利用的问题，Wang 等人[14]在 2018 年提出了一种名为 Revery 的自动化漏洞利用. Revery 利用模糊测试技术来寻找漏洞可利用状态的等效路径，并使用关键的内存操作指令来定位漏洞点. 随后，Revery 通过污点分析技术在等效路径中寻找可利用的状态. 最后运用符号执行技术求解路径约束和漏洞利用约束等条件，以生成漏洞利用方案.

3.7 其他可行的技术途径

还有一些自动漏洞利用工具未使用上述方法，此处进行集中论述，代表性工具有 !exploitable[23]，WAPTEC[25]，KernelPET[27]，BOPC[48]，SQLMi[51]，GuideExp[62]，Autosploit[57]，EXPRACE[66]等.

2009 年，微软公司基于其在 Vista 开发过程中使用模糊测试技术的经验开发了 !exploitable 工具[23]. 此工具旨在对崩溃现场自动进行分析和安全风险评估. 通过对比调用堆栈对应的散列值，该工具可以对崩溃情况进行分类，并通过分析崩溃上下文的语义信息来确定其可利用性级别.

2011 年，Bisht 等人[25]在 CCS 会议上发表了针对 Web 应用的白盒攻击工具 WAPTEC，首先使用约束引导搜索查找服务器控制路径，即导致敏感操作的路径；然后使用输入探测服务器检查结果控制路径上的敏感接收器，查找通向客户端拒绝的每个此类控制路径的输入.

2015 年李晓琦等人[27]针对 Linux 内核级提权漏洞提出了一种基于模拟攻击的漏洞检测思想，开发了漏洞自动利用系统 KernelPET. 该系统包括漏洞测试和一键提权 2 个部分. 漏洞测试部分能够根据用户需求，选择性地读取 Exploit 代码；而一键提权部分则通过自动化的策略式读取，能够快速定位当前平台上可利用的攻击代码.

2018 年由 Ispoglou 等人[48]提出了一种仅依赖数据的面向块编程的攻击方法 BOPC. 该方案将整个基本块视为程序代码的 Gadgets，在不触发控制流完整性等防护措施的条件下，构造实现任意内存写操作的方法. 该方法剔除了不可达路径以降低搜索空间，并利用启发式方法来指导搜索可行路径，最终构建了具备任意内存写能力的 Payload 集合.

2017 年 Jan 等人[51]提出了一个名为 SQLMi 的基于求解器和变异的测试生成框架，专门针对 XML 注入攻击. 该框架通过应用一系列变异操作符对普通的 XML 消息进行操作，可以支持 4 种类型的 XMLi 攻击样本的生成，最终绕过 XML 网关并对后端网络服务开展攻击.

2020 年北卡罗来纳大学的 Yilmaz 等人[62]针对 ActionScript 虚拟机不能利用模糊测试和符号执行来生成漏洞利用问题，开发了一个引导式的漏洞利用工具 GuideExp. GuidExp 探索触发切片执行之后的所有可能的执行路径，检查当前漏洞利用的子目标是否在任何执行路径中实现，并根据错误类型显示错误消息，以取消后续生成的候选切片的资格.

2020 年 Moscovich 等人[57]实现了一个评估漏洞可利用性的自动化框架 Autosploit, 其可代替漏扫工具和传统的手工或人工使用工具进行渗透测试. Autosploit 基于 2 种算法实现高效测试: 广义二值分裂和 Barinel, 分别用于无噪声和有噪声环境. Autosploit 能够在无噪声和有噪声环境中自动识别影响漏洞利用能力的系统属性.

2021 年 Wang 等人[73]提出了一种新型的自动化堆布局操作解决方案 MAZE. 其以概念验证(proof of concept, POC)样本、预期堆布局和漏洞程序作为输入, 首先提取可供用户操作堆的堆布局原语, 然后基于污点分析识别原语依赖关系, 使用符号执行分析原语语义, 最后通过改进的 Dig & Fill 算法组装堆布局原语. MAZE 将自动化堆布局问题转化为线性丢番图方程求解问题, 达到生成目标堆布局的原始堆操作序列的目的. 该方案可以生成复杂的堆布局并且适应主流的堆分配器, 辅助漏洞自动化利用方案生成. 但是该方案只支持特定的样本和特定的环境, 面对真实程序复杂的堆空间, 难以有效消除噪声生成准确的堆布局原语.

2021 年 Kang 等人[74]针对 JIT 的堆内存破坏漏洞提出了一个自动生成编译器漏洞利用代码的框架. 首先提取高层的漏洞利用原语, 如任意读、任意写、劫持控制流等; 然后在漏洞利用原语上构造出漏洞利用代码. 实现了对 JIT 生成漏洞利用代码, 并在 Javascript 引擎上实现了原型验证. 但部分环节仍停留在理论推导中, 缺乏实验数据的佐证, 对 JIT 漏洞利用的复杂性和保护绕过等需要进一步研究.

2021 年, 首尔大学的 Lee 等人[66]提出了一种针对内核非包含的多变量竞争的漏洞利用技术 EXPRACE, 首先划分易利用竞争和难利用竞争的内在条件, 然后在利用过程中操纵中断机制(如 rescheduling IPI, TLB shootdown IPI, membarrier IPI, hardware interrupts), 将难以利用的竞争转变为容易利用的竞争.

4 存在的不足及展望

综上分析可知, 当前漏洞自动化利用技术尽管整体呈现实用化的趋势, 但是该领域的一些根本性难题仍然未得到解决甚至缓解. 本文从现有成果与技术实用化所面临的差距方面, 总结 2 个方面的不足:

1)从共性技术方法的实际能力来看, 漏洞根源定位不准制约了全链自动化的实现. 如图 4 所示, 当前主流的漏洞自动化利用技术往往是基于异常点(crash point)的分析, 生成一条从异常点到利用点的路径(如 $B \rightarrow C \rightarrow D_1$). 但是实际应用时, 需要构造一条从漏洞根源到达攻击点的路径(如 $A \rightarrow B \rightarrow C \rightarrow D_1 \rightarrow E$), 难以实现全链的自动化利用. 导致这种现象的主要原因是: 程序深度理解、漏洞根源定位等基础性难题未得到有效解决.

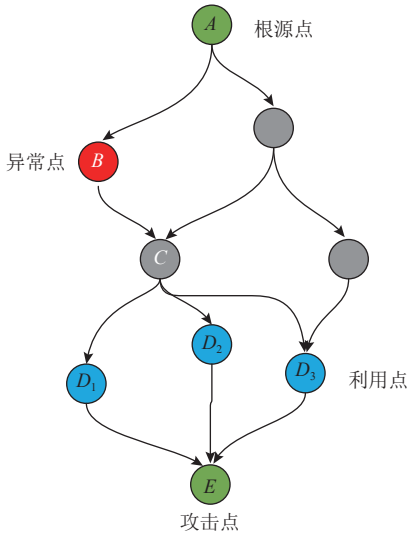

Fig. 4 Schematic diagram of path from vulnerability root to attack point

图 4 漏洞根源至攻击点路径示意图

2)从系统防御机制突破的自动化程度来看, 现有方法难以实现复杂防御机制的突破. 漏洞自动利用的本质是可利用路径下的复杂约束条件求解. 现有的自动化生成均是在受限路径上的探索和规划, 如现有的 CRAX[12], Revery[14]等工具均可实现 DEP 机制的绕过, 但是需要关闭 ASLR, CFI 等系统防御机制, 降低约束条件求解的难度. 在真实场景下, 该类“强约束”条件不仅无法满足, 往往更加苛刻, 如对漏洞利用代码的时效性等均有更高的要求. 使用现有基于异常点的漏洞自动化利用技术, “输入空间”受限, 难以求解出满足“强约束”条件的输入, 更无法实现复杂防御机制的突破.

因此, 从满足处理复杂结构软件对象, 以及实现复杂类型漏洞的自动化利用的目标来看, 未来需要重点围绕 2 个方面展开研究:

1)如何提高漏洞成因深度分析能力, 实现复杂场景下的漏洞可利用性判定. 现有研究成果难以实现漏洞利用过程的全链自动化, 其根本原因是无法准确定位漏洞触发的根源点(root-cause). 漏洞根源

直接反映了漏洞成因和漏洞的可利用性. 但是对于复杂结构软件、复杂类型漏洞，准确定位漏洞根源面临数据结构还原、代码控制流逆向等难题. 因此，以漏洞根源定位为核心的漏洞成因深度分析方向会是未来该领域研究的重点.

2)研究如何提高约束求解能力，实现非线性条件下的路径约束条件的准确求解. 现有研究成果难以实现复杂防御机制的突破，其中一个关键因素是约束求解能力有限. 结合符号执行和约束求解的漏洞自动化利用技术是目前最为主流的技术，但是受限于目前约束求解方法的算力，难以准确求解出非线性条件下的复杂约束. 因此，作为一个基础性难题，该问题的研究是无法绕过的一个研究方向.

5 总　　结

本文综述了漏洞自动化利用技术的研究成果，将漏洞自动化利用分为典型的 4 个步骤：漏洞根源定位、可达路径搜索、漏洞原语生成、利用代码生成. 从人机边界、攻防博弈、共性技术 3 个角度分别对当前代表性成果进行论述，阐述每个成果的技术思路和适用对象. 从技术实用化的角度，对当前研究总结了 2 个方面的不足，提出了 2 个研究方向. 漏洞自动化利用是漏洞领域研究中最具挑战性，也最能体现研究价值的方向，该领域新思路、新方法、新技术的出现，必将有力推动漏洞研究领域的进步.

作者贡献声明：武泽慧负责论文框架设计和核心内容撰写；魏强参与论文框架设计以及第 1 节和第 2 节内容撰写；王新蕾参与论文第 4 节内容撰写及全文规范性检查工作；王允超参与论文第 3 节内容撰写；燕宸毓参与全文的图片绘制、内容审校和修订；陈静负责正文和参考文献格式修订.

参　考　文　献

[1] Lu Hui, Jin Chengjie, Helu Xiaohan, et al. Research on intelligent detection of command level stack pollution for binary program analysis[J]. Mobile Networks and Applications, 2021(4): 1723−1732

[2] Hu Ning, Tian Zhihong, Lu Hui, et al. A multiple-kernel clustering based intrusion detection scheme for 5G and IoT networks[J]. International Journal of Machine Learning and Cybernetics, 2021, 12(11): 1−16

[3] Pan Menghai, Huang Weixiao, Li Yanhua, et al. DHPA: Dynamic human preference analytics framework: A case study on taxi drivers' learning curve analysis[J]. ACM Transactions on Intelligent Systems, 2020, 11(1): 1−19

[4] One A. Smashing the stack for fun and profit[J]. Phrack Magazine, 1996, 7(49): 14−16

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

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

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

[8] Wang Minghua, Su Purui, Li Qi, et al. Automatic polymorphic exploit generation for software vulnerabilities[C]//Proc of the 9th Int Conf on Security and Privacy in Communication Systems. Berlin: Springer, 2013: 216−233

[9] Avgerinos T, Cha S K, Rebert A, et al. Automatic exploit generation[J]. Communications of the ACM, 2014, 57(2): 74−84

[10] Hu Hong, Zheng Leongchua, Adrian S, et al. Automatic generation of data-oriented exploits[C]//Proc of the 24th USENIX Security Symp. Berkeley, CA: USENIX Association, 2015: 177−192

[11] 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

[12] 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 IEEE Int Conf on Software Security and Reliability. Piscataway, NJ: IEEE, 2012: 78−87

[13] He Liang, Cai Yan, Hu Hong, et al. Automatically assessing crashes from heap overflows[C]//Proc of the 32nd IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2017: 274−279

[14] 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. New York: ACM, 2018: 1914−1927

[15] Li Chao, Hu Jianwei, Cui Yanpeng. Automated exploitation of buffer overflow vulnerabilities based on symbolic execution[J]. Computer Application and Software, 2019, 36(9): 327−333(in Chinese)
(李超，胡建伟，崔艳鹏. 基于符号执行的缓冲区溢出漏洞自动化利用[J]. 计算机应用与软件，2019，36(9)：327−333)

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

[17] Chen Yueqi, Lin Zhenpeng, Xing Xinyu. A systematic study of elastic objects in kernel exploitation[C]//Proc of the 2020 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2020: 1165−1184

[18] Liguori P, Al-Hossami E, Cotroneo D, et al. Shellcode_IA32: A

dataset for automatic shellcode generation[J]. arXiv preprint, arXiv: 2104.13100, 2021

[19] Yang Guang, Chen Xiang, Zhou Yanlin, et al. DualSC: Automatic generation and summarization of shellcode via transformer and dual learning[C]//Proc of the 29th IEEE Int Conf on Software Analysis Evolution and Reengineering (SAER 2022). Piscataway, NJ: IEEE, 2022: 361−372

[20] Lu Hui, Jin Chengjie, Helu Xiaohan, et al. AutoD: Intelligent blockchain application unpacking based on JNI layer deception call[J]. IEEE Network, 2020, 35(2): 215−221

[21] Khandait P, Hubballi N, Mazumdar B. IoTHunter: IoT network traffic classification using device specific keywords[J]. IET Networks, 2021, 10(2): 59−75

[22] 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

[23] MICROFOST. The history of the !exploitable crash analyzer[EB/OL]. 2009[2023-10-15]. http://blogs.technet.com/b/srd/archive/2009/04/08/the-history-of-the-exploitable-crash-analyzer/

[24] Miller C, Caballero J, Johnson N M, et al. Crash analysis with BitBlaze[J/OL].2010[2022-08-07].https://media.blackhat.com/bh-us-10/whitepapers/Miller/BlackHat-USA-2010-CMiller-BitBlaze-wp.pdf

[25] 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. New York: ACM, 2011: 575−586

[26] Huang S K, Lu H L, 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

[27] Li Xiaoqi, Liu Qixu, Zhang Yuqing. An automatic exploit system for kernel power lifting vulnerabilities based on simulated attacks[J]. Journal of the University of Chinese Academy of Sciences, 2015, 32(3): 384−390 (in Chinese)
(李晓琦, 刘奇旭, 张玉清. 基于模拟攻击的内核提权漏洞自动利用系统[J].中国科学院大学学报, 2015, 32(3): 384-390)

[28] Zhang Puhan, Wu Jianxiong, Wang Xin, et al. Program crash analysis based on taint analysis[C]//Proc of the 9th Int Conf on P2P, Parallel, Grid, Cloud and Internet Computing. Piscataway, NJ: IEEE, 2014: 492−498

[29] Do Q H, Bubel R, Hähnle R. Exploit generation for information flow leaks in object-oriented programs[C]//Proc of the 30th Int Information Security and Privacy Conf. Berlin: Springer, 2015: 401−415

[30] Ruffell M, Hong J B, Kim H, et al. Towards automated exploit generation for embedded systems[C]//Proc of the 17th Int Workshop on Information Security Applications. Berlin: Springer, 2016: 161−173

[31] 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. New York: ACM, 2016: 641−652

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

[33] Shoshitaishvili Y, Wang Ruoyu, Salls C, et al. Sok: State of the art of war: Offensive techniques in binary analysis[C]//Proc of the 2016 IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2016: 138−157

[34] Luo Lannan, Zeng Qiang, Cao Chen, 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. New York: ACM, 2017: 225−238

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

[36] Brooks T N. Survey of automated vulnerability detection and exploit generation techniques in cyber reasoning systems[C]//Proc the 2018 Science and Information Conf. Berlin: Springer, 2018: 1083−1102

[37] Wan Yunpeng, Deng Yi, Shi Donghui, et al. Automatic utilization 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)

[38] 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. New York: ACM, 2017: 661−671

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

[40] 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: 125−136

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

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

[43] 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

[44] 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. Piscataway, NJ: IEEE, 2018: 705−712

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

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

[47] Garmany B, Stoffel M, Gawlik R, et al. Towards automated generation of exploitation primitives for web browsers[C]//Proc of the 34th Annual Computer Security Applications Conf. New York: ACM, 2018: 300−312

[48] Ispoglou K K, AlBassam B, Jaeger T, et al. Block oriented programming: Automating data-only attacks[C]//Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2018: 1868−1882

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

[50] Huang Ning, Huang Shuguang, Chang Chao. Analysis to heap overflow exploit in Linux with symbolic execution[C]//Proc of the 2019 IOP Conf Series: Earth and Environmental Science. Bristol: IOP Publishing, 2019: 4−20

[51] 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, 2017, 45(4): 335−362

[52] Fang Hao, Wu Lifa, Wu Zhiyong. A symbolic execution-based return-to-dl-resolve approach for automatic code generation using[J]. Computer Science, 2019, 46(2): 127−132 (in Chinese)
(方皓, 吴礼发, 吴志勇. 基于符号执行的 return-to-dl-resolve 利用代码自动生成方法[J]. 计算机科学, 2019, 46(2): 127−132)

[53] Chen Yueqi, Xing Xinyu. Slake: Facilitating slab manipulation for exploiting vulnerabilities in the Linux kernel[C]//Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 1707−1722

[54] Wu Wei, Chen Yueqi, Xing Xinyu, et al. KEPLER: Facilitating control-flow hijacking primitive evaluation for Linux kernel vulnerabilities[C]//Proc of the 28th USENIX Security Symp. Berkeley, CA: USENIX Association, 2019: 1187−1204

[55] Heelan S, Melham T, Kroening D. Gollum: Modular and greybox exploit generation for heap overflows in interpreters[C]//Proc of the 2019 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2019: 1689−1706

[56] Wei Yuan, Luo Senlin, Zhuge Jianwei, et al. ARG: Automatic ROP chains generation[J]. IEEE Access, 2019, 7: 120152−120163

[57] Moscovich N, Bitton R, Mallah Y, et al. Autosploit: A fully automated framework for evaluating the exploitability of security vulnerabilities[J]. arXiv preprint, arXiv: 2007.00059, 2020

[58] Schwartz E J, Cohen C F, Gennari J S, et al. A generic technique for automatically finding defense-aware code reuse attacks[C]//Proc of the 2020 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2020: 1789−1801

[59] Chen Weiteng, Zou Xiaochen, Li Guoren, et al. KOOBE: Towards facilitating exploit generation of kernel out-of-bounds write vulnerabilities[C]//Proc of the 29th USENIX Security Symp. Berkeley, CA: USENIX Association, 2020: 1093−1110

[60] Chae S, Jin Hongjoo, Park M C, et al. HS-Pilot: Heap security evaluation tool model based on atomic heap interaction[J]. IEEE Access, 2020, 8: 201914−201924

[61] Zhao Zixuan, Wang Yan, Gong Xiaorui. HAEPG: An automatic multi-hop exploitation generation framework[C]//Proc the 17th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2020: 89−109

[62] Yilmaz F, Sridhar M, Choi W. Guide me to exploit: Assisted ROP exploit generation for actionscript virtual machine[C]//Proc of the 2020 Annual Computer Security Applications Conf. New York: ACM, 2020: 386−400

[63] Yun I, Kapil D, Kim T. Automatic techniques to systematically discover new heap exploitation primitives[C]//Proc of the 29th USENIX Security Symp. Berkeley, CA: USENIX Association, 2020: 1111−1128

[64] Blazytko T, Schlögel M, Aschermann C, et al. AURORA: Statistical crash analysis for automated root cause explanation[C]//Proc of the 29th USENIX Security Symp. Berkeley, CA: USENIX Association, 2020: 235−252

[65] Deng Fenglei, Wang Jian, Zhang Bin, et al. A pattern-based software testing framework for exploitability evaluation of metadata corruption vulnerabilities[J]. Scientific Programming, 2020, 5(10): 1−21

[66] Lee Y, Min C, Lee B. EXPRACE: Exploiting kernel races through raising interrupts[C]//Proc of the 30th USENIX Security Symp. Berkeley, CA: USENIX Association, 2021: 2363−2380

[67] Ma Xiangning, Zhang Zhaoqing, Feng Xiaobing, et al. Process recovery techniques in binary translation[J]. Computer Engineering and Applications, 2002, 38(19): 1−5 (in Chinese)
(马湘宁, 张兆庆, 冯晓兵, 等. 二进制翻译中的过程恢复技术[J]. 计算机工程与应用, 2002, 38(19): 1−5)

[68] Fu Wen, Wei Bo, Zhang Tianlei, et al. Application and implementation of process recovery techniques in IA64 binary translation[J]. Computer Engineering and Applications, 2006, 42(21): 81−83 (in Chinese)
(付文, 魏博, 张天雷, 等. 过程恢复技术在 IA64 二进制翻译中的应用与实现[J]. 计算机工程与应用, 2006, 42(21): 81−83)

[69] Fang Xia, Yin Qing, Jiang Liehui, et al. A data flow analysis based register parameter recovery method[J]. Computer Engineering, 2009, 35(22): 62−64 (in Chinese)
(方霞, 尹青, 蒋烈辉, 等. 基于数据流分析的寄存器参数恢复方法[J]. 计算机工程, 2009, 35(22): 62−64)

[70] Wang Ruipeng, Pan Zulie, Shi Fan, et al. AEMB: An automated exploit mitigation bypassing solution[J]. Applied Sciences, 2021, 11(20): 9727−9728

[71] Ispoglou K K, Austin D, Mohan V, et al. FuzzGen: Automatic Fuzzer

generation[C]//Proc of the 29th USENIX Conf on Security Symp. Berkeley, CA: USENIX Association, 2020: 2271−2287

[72] Zhang Hangwei, Lu Kai, Zhou Xu, et al. SIoTFuzzer: Fuzzing Web interface in IoT firmware via stateful message generation[J]. Applied Sciences, 2021, 11(7): 3120−3121

[73] Wang Yan, Zhang Chao, Zhao Zixuan, et al. MAZE: Towards automated heap feng shui[C]//Proc of the 2021 USENIX Security Symp. Berkeley, CA: USENIX Association, 2021: 1647−1664

[74] Kang Xiyu, Debray S. A framework for automatic exploit generation for JIT compilers[C]//Proc of the 2021 Research on Offensive and Defensive Techniques in the Context of Man at the End (MATE) Attacks. New York: ACM, 2021: 11−19

Wu Zehui, born in 1988. PhD, lecturer. His mian research interest includes software and system vulnerability analysis.

武泽慧, 1988 年生. 博士, 讲师. 主要研究方向为软件与系统漏洞分析.

Wei Qiang, born in 1979. PhD, professor, PhD supervisor. His main research interests include software security analysis and industrial control system security.

魏　强, 1979 年生. 博士, 教授, 博士生导师. 主要研究方向为软件安全分析、工控系统安全.

Wang Xinlei, born in 1990. Bachelor, assistant engineer. Her main research interest includes software security analysis.

王新蕾, 1990 年生. 学士, 助理工程师. 主要研究方向为软件安全分析.

Wang Yunchao, born in 1992. PhD, lecturer. His main research interest includes software and system vulnerability analysis.

王允超, 1992 年生. 博士, 讲师. 主要研究方向为软件与系统漏洞分析.

Yan Chenyu, born in 1996. Master, teaching assistant. Her main research interest includes software security analysis.

燕宸毓, 1996 年生. 硕士, 助教. 主要研究方向为软件安全分析.

Chen Jing, born in 1999. Master. Her main research interest includes software security analysis.

陈　静, 1999 年生. 硕士. 主要研究方向为软件安全分析.