文章编号:1671-4229(2019)03-0052-07

软件漏洞自动利用研究综述

苏璞睿[1,2]，黄桦烽[1,2]，余媛萍[1,2]，张　涛[3]
(1. 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；
2. 中国科学院大学 计算机科学与技术学院，北京　100190；3. 中国信息安全测评中心，北京　100085)

摘　要：近年来,软件漏洞已成为系统安全与攻防对抗的核心要素,随着软件数量的增加和规模的复杂化,漏洞数量逐年增加,而依赖于人工的漏洞分析与利用生成已难以满足现实需求,漏洞的自动分析和利用生成是亟待解决的难点问题. 现有研究已经取得了相关的成果,文章从控制流劫持漏洞自动利用、面向堆漏洞的自动分析与利用、安全机制自动化对抗方法和综合性的漏洞自动利用框架等四个方面介绍当前软件漏洞自动利用研究进展,进而分析未来软件漏洞自动利用发展趋势.

关键词：漏洞；控制流劫持；安全机制；漏洞利用自动生成
中图分类号：TP 311　　**文献标志码**：A

软件漏洞是存在于系统或者应用软件中,可导致攻击者在未授权情况下访问或破坏系统的安全缺陷. 软件漏洞形成的原因具有多样性,包括开发人员疏忽及水平受限、编译器安全缺陷、程序功能逻辑复杂、程序测试不充分等因素. 鉴于软件的高复杂度,包括代码规模、功能组成、多线程并发、数据资源共享等复杂机制,即使经过代码审查与测试也难以消除软件中的安全漏洞,以致严重威胁到系统的信息系统的安全.

软件漏洞是网络安全的主要威胁,随着软件数量的和规模的扩张,漏洞数量逐年增加,而依赖于人工的漏洞分析与利用生成已难以满足现实需求. 同时,漏洞利用是系统入侵渗透的主要手段,如何提高软件漏洞的自动化利用能力是攻防双方共同关注的焦点. 随着基础安全分析技术的发展和漏洞分析需求的增加,人类逐步推动漏洞自动攻防的研究. 在漏洞自动分析与利用所需技术基本满足要求的情况下,2014 – 2016 年美国 DARPA 组织的 CGC(Cyber Grand Challenge)比赛在漏洞自动攻防方向进行了初步的尝试,并受到业界的广泛关注[1]. CGC 是实现漏洞挖掘、分析、利用、修复等完全自动化的一次尝试,参赛团队通过建立“自动攻击防御系统”,在无人干预的条件下,自动寻找程序漏洞、生成利用攻击敌方,以及部署补丁抵御对手的攻击. CGC 是软件漏洞挖掘与自动化利用的里程碑,国内从 2017 年开始组织类似的自动攻防比赛 RHG(Robot Hacking Game),验证了无人干预条件下的自动化网络攻防的可行性,构建出一套具备自主攻击及自我防御的软件,推动了软件漏洞自动利用的发展.

本文根据漏洞自动利用各环节面临的不同问题,利用方法及技术的差异进行分类总结,从控制流劫持漏洞自动利用、面向堆漏洞的自动分析与利用、安全机制的自动化对抗方法和综合性的漏洞自动利用框架等四个方面,介绍当前软件漏洞自动利用研究的进展,从而分析未来软件漏洞自动利用发展趋势.

1　控制流劫持漏洞自动利用方法

漏洞利用的主要目的是在程序中执行满足攻击者意图的代码,如下载恶意代码、执行 shell、添加账户等功能,也就是通常所说的任意代码执行漏洞. 而控制流劫持漏洞就是一种典型的任意代码执行漏洞,这种漏洞大多属于高危漏洞,针对这种漏洞自动化地生成利用代码已有多年的研究.

收稿日期：2019 – 04 – 21；　**修回日期**：2019 – 05 – 12
基金项目：国家自然科学基金资助项目(U1736209,61572483,U1836117,U1836113)
作者简介：苏璞睿(1976—),男,研究员. E-mail:purui@ iscas. ac. cn

控制流劫持漏洞自动利用构造的基本思路是在程序动态运行过程中基于数据流分析方法,检测程序是否运行到一个执行指令可被输入控制的状态,这些状态包括:指令代码受到输入控制、指令指针寄存器直接或间接受到输入代码控制;执行了可直接或者间接引起任意代码执行的相关函数,如 system 等命令执行函数参数可任意控制. 当程序执行到执行指令可被输入控制的状态时,基于当前的输入数据为模板,关联输入与执行路径直接的约束关系,同时建立劫持点、攻击载荷(shellcode)与输入数据的依赖关系,最后基于符号执行等约束求解方法自动生成漏洞利用攻击代码. 相关工作包括 APEG[2]、AEG[3-4]、Mayhem[5]、CRAX[6]、PolyAEG[7],本文根据其各自特点分别展开分析.

1.1　补丁信息辅助的漏洞自动利用构造

基于补丁信息的漏洞自动利用构造核心思想是针对补丁位置的过滤约束条件,基于污点分析和符号执行的基本方法构造违反补丁约束条件的输入,最后基于污点分析判断违规输入是否造成控制流劫持,进一步构造满足漏洞利用条件. APEG 是在 2008 年,Brumley 等[2]提出的基于二进制补丁对比的程序漏洞自动利用方法. 此方法借助了补丁信息对比的辅助,也是漏洞自动利用的首次尝试,其关注重点其实是基于补丁对比实现漏洞挖掘,由于早期的字符串拷贝溢出漏洞泛滥等因素,该方法对微软的多个补丁程序进行测试并验证了方法的有效性. 但实际效果表面,其在漏洞利用代码生成方面并未深入,所构造的利用大多只是崩溃,未能通过控制流劫持达到任意代码执行的效果.

1.2　源码辅助的漏洞自动利用构造

源码辅助的漏洞自动利用构造基本思想是基于源码编译过程中的中间信息,数据变量及代码逻辑依赖关系,提取程序中的脆弱点和触发脆弱点的约束条件,构造潜在威胁的输入,再结合二进制程序动态运行的信息,自动根据相关约束条件构造漏洞利用代码并进行验证. AEG 是 2011 年 Avgerinos 等[3-4]在 NDSS 会议上提出的漏洞自动挖掘与利用方法. 此方法借助了源代码的辅助,提取编译过程中的 LLVM 字节码文件,通过字节码文件的分析提取漏洞触发路径约束,以及漏洞相应的内存变量属性,通过约束求解获得触发漏洞的 POC 输入;然后将 POC 输入载入程序动态执行,通过程序动态指令插桩技术,基于数据流分析内存布局关系,建立攻击代码布局约束条件,使用优化的混合符号执行求解路径约束与攻击代码布局约束条件,实现了漏洞利用自动生成. AEG 依赖于程序源代码进行程序错误静态搜索和路径约束条件的生成,同时,其只支持栈溢出和格式化字符串漏洞的利用自动生成.

1.3　针对二进制代码的漏洞自动利用构造

针对二进制代码的漏洞自动利用构造其核心思想是结合二进制代码静动态分析、污点传播和符号执行方法,构造触发程序执行脆弱点的 POC,再基于插桩分析监测 POC 对目标程序破坏程度的影响,通过内存污点数据布局分析、二进制代码执行路径约束条件的求解自动构造漏洞利用代码.

Mayhem 是 2012 年 Cha 等[5]在 S&P 会议上提出的针对二进制程序的漏洞利用自动生成方法. 该方法综合利用了离线式符号执行内存消耗低的特点和在线式符号执行速度快的优势,构建出基于索引的内存模型,用于优化处理符号化内存的加载问题,以提高符号执行的效率. 在程序运行状态检测方面,Maythem 通过插桩分析和污点传播检测 call/jmp/ret 等敏感指令是否受到输入控制,当检测出程序运行到控制流劫持状态后,将相应的输入信息提交给符号执行子系统进行分析和利用代码自动构造. CRAX 是 2012 年 Huang 等[6]提出的基于 S2E 符号执行平台,对触发程序崩溃的 POC 进行动态分析,优化过滤了无关函数,提升了约束求解效率,支持格式化字符串、堆栈溢出、变量未初始化等多种类型漏洞,但因采用了重量级的符号执行引擎 S2E,其效率及资源消耗难以满足实际的应用需求.

随着攻防双方对抗过程的发展,可利用漏洞资源是相对稀缺的,攻击者为对抗病毒特征码扫描需要对漏洞攻击代码进行变形,而防御者为了能够检测变形的攻击代码也需要主动构造变形的攻击代码测试和提升检测引擎的能力,因此,漏洞利用样本的多样性生成对于漏洞攻击与防御双方能力的提升具有重要的促进作用. 漏洞利用样本多样性生成的基本思想是基于动态污点分析检测所有的程序控制流劫持点,搜索潜在的攻击路径,通过攻击代码分段布局、等效指令替换等手段构建不同的控制流转移模式,实现漏洞利用样本的

多样性生成. Wang 等[7]在 2013 年的 Secure Comm 会议上提出了一套针对控制流劫持类漏洞的利用样本多样性自动生成方法 PolyAEG. PolyAEG 在实现上通过 QEMU 进行二进制指令代码插桩,通过反汇编指令的分析实现污点传播,通过树结构记录污点传播流图和压缩 MAP 记录内存和寄存器存储单元污点状态. PolyAEG 针对覆盖返回地址和覆盖函数指针的控制流劫持漏洞实现了一套漏洞利用样本多样化生成的自动化构造方案,测试验证了 8 个现实漏洞程序,其中针对单个控制流劫持漏洞最多生成了 4 724 个利用样本变种. 该方案是一种较为有效的控制流劫持漏洞自动利用生成方法,但对于尚未触发控制流劫持点的 POC 样本难以生成有效利用,对 POC 具有较强依赖性.

2 面向堆漏洞的自动分析与利用

堆漏洞是一种常见的漏洞类型,包括堆溢出、释放后重用(Use After Free 简称 UAF)、Off by One 等. 与栈溢出等控制流劫持漏洞相比,堆漏洞的利用过程更为复杂,这主要是由于堆数据类型的多样性导致,不像栈存在一个返回地址指针,堆数据的多样性体现在堆数据是动态生成的,可以是一个整型变量、一块缓冲区、一个地址指针,甚至是一个函数指针等. 而堆漏洞利用的基本思路是破坏堆结构中的地址指针或者函数指针,来实现任意地址写的能力或控制流劫持. 堆结构的复杂多样性和 safe unlink 等机制给堆漏洞自动利用带来了挑战,模糊测试等方法挖掘出的堆漏洞 POC 大多只是一个崩溃,尚未达到任意地址写或者控制流劫持的能力. 近年来,安全社区和学术界共同致力于推进漏洞自动分析与利用技术的研究,并且取得了相关进展.

漏洞的可利用性分析是漏洞利用自动构造的前提,堆漏洞与控制流劫持漏洞最大的差异就是其可利用性无法保证. 堆漏洞的可利用性判定主要思路是分析堆漏洞所能破坏的数据内容,进一步分析破坏的数据内容是否可操控,以及这部分内容将对程序后续执行造成怎样的影响. 基于该思路,He 等[8]提出了基于堆溢出崩溃和溢出修复的堆漏洞可利用性评估方法 HCS IFTER,该方法首次提出了通过修复被破坏的内存,使得程序能够继续执行,并检测被破坏的内存数据后续使用情况,进而分析判定内存破坏点是否潜在利用条件,虽然该方法能够检测出触发崩溃样本本身是否潜在可利用性,但还不具备内存重新布局和构造的能力,即使检测出当前条件不可利用也不能得出该漏洞不可利用的结论.

针对堆破坏相关内存重新布局和构造能力提升,以触发更多执行路径及状态的需求,Wang 等[9]在 2018 年提出了一种基于已知崩溃路径,结合导向模糊测试的技术,获取漏洞程序更多可利用状态的方法 Revery. 该方法定义了两种可利用的状态,包括任意内存写和任意地址执行. 首先,使用动态分析研究崩溃路径下漏洞的细节,从而构建漏洞内存布局与其贡献指令的关系图;然后,运用内存布局导向的模糊测试搜索可利用路径;最后,基于控制流缝合技术将崩溃路径和可利用路径缝合生成能够同时触发漏洞和达到可利用状态的利用样本. Revery 是基于 ANGR 开源框架实现的,针对 19 个 CTF 样本进行了测试验证,其中 47% 可以直接生成利用代码,26% 可以触发到可利用状态.

堆漏洞利用自动构造难点在于堆数据结构的多样性,解决这一问题堆思路是将数据流分析与模糊测试方法相结合. 国外的 Repel 等[10]针对堆漏洞自动分析与利用困难点在于需构造满足堆操作函数的参数和特定的堆操作序列,构建了一个可利用状态堆操作序列数据库,在堆操作函数模糊测试过程中挑选出堆操作序列匹配成功的测试用例,并设计了一个原型系统验证了该方法,但是产生合适的测试用例效率低. 针对效率问题,Heelan 等[11]提出通过构造输入自动操控堆内存布局实现堆漏洞利用的方法,该方法解决了上面方法产生合适测试用例带来的时间开销,但是堆布局成功的稳定性与脆弱点的堆大小和构造利用的数据结构有关. Yun 等[12]针对堆开发技术的根源问题,通过找出像 jemalloc、ptmalloc、tcmalloc 这些堆管理函数的共性,从而更好地辅助堆漏洞利用自动构造.

UAF 漏洞是堆漏洞中的常见漏洞,广泛出现在浏览器和操作系统内核中. 宾州州立大学的 Wu 等[13]在 2018 年的 USENIX Security 会议上提出了针对内核 UAF 漏洞的自动利用框架 FUZE,该框架利用内核模糊测试技术提供更多的内核崩溃上下文环境作为漏洞利用的依据,并利用符号执行

求解在不同的上下文环境中去尝试利用目标漏洞,选取了 15 个内核 Use-After-Free 漏洞进行了验证,其中 12 个漏洞能实现利用,而公开的资料只有 5 个是被黑客成功利用的,提升了内核 UAF 漏洞利用的能力.

3　安全机制自动化对抗方法

软件安全机制缓解技术可以在一定程度上缓解漏洞利用攻击,典型的安全机制包括 Stack Guard[14]、DEP (Data Execution Prevention)[15]、ASLR(Address Space Layout Randomization)[16] 和 CFI(Control-Flow Integrity)[17]. 安全机制的自动化对抗研究,是软件漏洞自动利用过程中的一个重要环节,国内外研究团队从 ROP、DOP 和 BOP 等方面给出了安全机制自动绕过的相关方案.

针对数据执行保护 DEP 的利用缓解方案,攻击者通常采用 ROP/JOP 等方式实现漏洞利用代码,其基本思想是基于目标程序自身代码片段,收集其中特定功能片段的 Gadget, 并通过 Gadget 的自动化编程拼接构建 ROP,生成满足特定功能的代码序列. 针对 ASLR 通常需要未开启 ASLR 的相关模块或者信息泄露漏洞. 典型的代表工作是 2011 年 Schwartz 等[18] 在 USENIX Security 会议上提出的一套面向高可靠性漏洞利用的 ROP 代码自动生成方法 Q. 该方法首先搜索系统中未使用地址随机化的程序或动态库模块,在这些模块中找出具备特定功能的 Gadget 集合,同时提供了一种称为 QooL 的编程语言,将满足特定语义功能的目标代码编译成面向 Gadget 的指令序列,最后通过利用已获取的 Gadget 集合及对应地址,生成 Gadget 片段对应的最终 ROP 代码. 该方法通过对 9 个真实软件漏洞的实验,验证了数据执行保护和地址随机化功能可被绕过的结论,但其依赖于系统中存在未随机化模块的前提,在早期 ASLR 还未大范围普及的情况下能发挥一定作用,但在当前新版本系统及软件环境下效果欠佳.

国内的方皓等提出了基于符号执行与安全机制绕过技术 Return-to-dl-resolve 结合的自动利用代码生成方案 R2dlAEG[19-20],通过控制流劫持漏洞利用基本方法与 Return-to-dl-resolve 过程自动化实现的结合,针对控制流劫持漏洞,实现了自动生成能绕过 ASLR 与 NX 防护机制的利用代码,同时不依赖于未随机化的动态库模块和信息泄露漏洞,并通过 CTF 样本和部分已知漏洞进行了验证.

CFI 是一种控制流完整性的漏洞利用缓解技术,逐步在 Windows10 等新版本系统得到应用,用于检测和拦截控制流劫持漏洞利用. 为了对抗 CFI 的防护机制,面向数据流攻击的利用方法应运而生,其思想在不直接影响和操控程序控制流的前提下,通过分析程序对输入数据的处理,构造出一个或多个利用输入数据改变程序数据流的利用路径,进而完成权限提升、认证机制绕过和信息泄露等功能. 在数据执行保护、地址随机化以及控制流完整性等安全机制防护下,面向数据流的利用方法显得熠熠生辉.

面向数据流攻击的代表性工作是 Hu 等[21] 在 2015 年提出的 Flow Stitch,该方法利用已知内存错误直接或间接地篡改程序原有数据流中的关键变量,通过比对错误执行记录和正常执行记录,筛选并确定内存错误影响范围中的敏感数据,来完成信息泄露、权限提升等利用的构造. 2016 年 Hu 等[22] 继而提出 DOP,即基于数据流的攻击代码编程模型,同时给出针对实际应用程序的基于数据流的攻击代码块和指令调度分配代码块的搜索、提取和编程方法,初步显示 DOP 是图灵完备的,能够实现任意代码的执行,并能绕过 DEP、ASLR 和 CFI 等系统防护措施.

针对 CFI 的对抗,除了数据流攻击还要面向块编程代码重用技术 BOP,该方法是在 2018 年由 Ispoglou 等[23] 提出的只依赖于数据的面向块编程的攻击方案 BOPC. BOPC 是一种自动评估攻击者能否在使用了 CFI 防御的二进制程序上实现任意代码执行的方案. 其主要思路是引入一种面向块编程的代码重用技术,它利用整个基本块作为程序代码 Gadgets,在不违反 CFI 等防护策略情况下编织实现任意内存写原语,并通过修剪不可达路径来减少搜索空间,使用启发式方法引导搜索可行路径,构建任意内存写能力的 payload 集合.

4　综合性的漏洞自动利用框架

软件漏洞自动利用是一系列复杂过程的组合,包括最基础的栈溢出利用[24],大多基于动态分析展开,在程序动态执行过程中监测程序输入数据,对其进行污点源标记,基于指令集的代码插桩

技术提取所执行的每一条指令,再通过反汇编或者中间语言解析指令的语义实现污点传播分析及符号执行表达式构建,然后在敏感指令等位置检测是否为可利用漏洞,当检测为可利用点之后,基于约束求解和内存布局分析,自动生成漏洞利用代码. 整个漏洞自动利用过程所依赖的基础方法主要包括指令代码插桩、指令分析、污点传播分析和符号执行,ANGR 就是利用这些技术形成的代表性的综合性漏洞自动利用开源框架,可自行源码安装,也可以使用其 docker 虚拟机[25].

ANGR 是由加州大学圣塔芭芭拉分校的 Shoshitaishvili 等[25]提出的面向二进制软件攻防的基础分析框架平台. 该框架主要是将已有基础性软件分析方法进行模块化实现,形成一套功能完整的分析框架,包括控制流图生成、动态分析执行、符号执行、值域分析等内容. ANGR 的指令插桩基于 QEMU 实现,类似的工具还有 PIN[26-27]、DynamoRIO[28]等;ANGR 的指令反汇编解析基于 Capstone[29]反汇编引擎,同类工具还有 udis86[30]等;ANGR 的符号执行求解基于 Z3[31]实现,类似的引擎还有 S2E[32]等.

ANGR 是一个程序漏洞分析与利用的基础框架,在此基础上可以实现针对二进制程序的漏洞挖掘、分析与利用等具体功能,同时可以将各项功能组合构建成自动化的漏洞分析利用系统. ANGR 的作者将该系统应用到了 CGC 比赛中,通过组合不同的分析技术与方法构建面向二进制软件的漏洞自动挖掘与利用系统. ANGR 本身也提供了程序漏洞利用生成的 AEG 插件,虽然针对其测试用例能够成功生成利用样本,但针对静态链接等规模稍微大一点的程序,会产生内存爆炸导致无法成功生产利用代码.

ANGR 被广泛应用到 CTF 比赛中进行二进制代码分析和漏洞利用代码生成,GitHub 中有一个 angr_ctf 的项目集成了 18 个左右的插件用于 CTF 样本分析,功能包括程序及库文件分析,任意地址读、写、跳转,API 劫持,寄存器、内存、栈的符号化分析等功能. 另外,学术界也有不少基于 ANGR 平台开展的漏洞自动挖掘与利用研究工作[9,33-34],其更侧重于验证方法的可行性,测试用例大多选择简单的 CTF 样本,距离真正的实用还有一定距离.

5 总结与展望

综上所述,目前漏洞自动利用研究较为成熟的主要针对控制流劫持漏洞,随着漏洞利用缓解机制的发展,安全机制的自动化对抗技术也逐步发展,而堆漏洞是比较复杂的漏洞类型,其利用过程需逐步构造突破任意地址写到任意地址执行,仍有较大的突破空间. 而目前尚未出现实用性的漏洞自动利用系统,大部分工作都只是一个原型系统,包括 ANGR 框架,其自动化程度也不高,需要针对特定样本,结合人工分析进行调整才能真正发挥效果.

针对特定目标的漏洞利用自动生成可能是未来的一个趋势,包括针对 Linux 内核提权[35]、针对浏览器[36]、针对固件设备[37]和针对 Android 系统组件[38]等. 由于特定目标具有其特定的属性,脆弱环节和漏洞利用方式存在一些特点,基于这些特征知识可内置特定的利用环节辅助漏洞利用的快速生成.

现有的漏洞利用自动生成大多基于模糊测试、污点分析、符号执行求解这些专家知识与漏洞利用技巧融合实现,还不具备真正的 AI 智能. 目前 AI 领域主要应用在分类、判断、策略择优等方面,其具有一定的概率性. 而漏洞利用需要具备相应的知识基础,过程步骤复杂,相关步骤需要精确计算,这些步骤并不适合使用 AI 智能技术. 未来 AI 智能技术在漏洞自动利用的应用上,还需要与专家系统知识相结合,在部分不依赖于精确计算的环节上辅助提升漏洞利用自动生成的能力,如多路径情况下利用 AI 技术进行路径选择与择优.

参考文献:

[1] Song J, Alvesfoss J. The DARPA cyber grand challenge: A competitor's perspective, part 2[J]. IEEE Security & Privacy, 2015, 13(6):72-76.

[2] Brumley D, Poosankam P, Song D, et al. Automatic patch-based exploit generation is possible: Techniques and implications[C]//Security and Privacy, 2008. SP 2008. IEEE Symposium on. Los Alamitos: IEEE, 2008:143-157.

[3] Avgerinos T, Sang K C, Hao B L T, et al. AEG: Automatic exploit generation[C]//Network and Distributed System Secu-

rity Symposium, NDSS 2011. San Diego: DBLP, 2011:1-18.

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

[5] Cha S K, Avgerinos T, Rebert A, et al. Unleashing mayhem on binary code[C]//Proceeding of the 2012 IEEE Symposium on Security and Privacy. Los Alamitos: IEEE, 2012: 380-394.

[6] 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]//Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on. Los Alamitos: IEEE, 2012.

[7] Wang M, Su P, Li Q, et al. Automatic polymorphic exploit generation for software vulnerabilities[C]//International Conference on Security and Privacy in Communication Systems. Berlin: Springer, 2013:216-233.

[8] He L, Cai Y, Hu H, et al. Automatically assessing crashes from heap overflows[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. Los Alamitos: IEEE, 2017: 274-279.

[9] Wang Y, Zhang C, Xiang X, et al. Revery: From proof-of-concept to exploitable[C]//Computer and Communications security. New York: ACM, 2018: 1914-1927.

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

[11] Heelan S, Melham T, Kroening D, et al. Automatic heap layout manipulation for exploitation[C]//27th USENIX Security Symposium. Berkeley: USENIX, 2018: 763-779.

[12] Yun I, Kapil D, Kim T, et al. Automatic techniques to systematically discover new heap exploitation primitives[J/OL]. [2019-04-29]. ArXiv: Cryptography and Security, 2019, doi:abs1903.00503:1-17.

[13] Wu W, Chen Y, Xu J, et al. FUZE: Towards facilitating exploit generation for kernel use-after-free vulnerabilities[C]//27th USENIX Security Symposium (USENIX Security 18). Berkeley: USENIX, 2018: 781-797.

[14] Cowan C, Pu C, Maier D, et al. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks[C]//Conference on Usenix Security Symposium. Berkeley: USENIX, 1998:63-78.

[15] Microsoft Corporation. Data execution prevention[EB/OL]. (2009-02-22)[2019-04-20]. http://technet.microsoft.com/en-us/library/cc738483.aspx.

[16] Shacham H, Page M, Pfaff B, et al. On the effectiveness of address-space randomization[C]//Proceedings of the 11th ACM Conference on Computer and Communications Security. New York: ACM, 2004: 298-307.

[17] Abadi M, Budiu M, Erlingsson U, et al. Control-flow integrity[C]//Proceedings of the 12th ACM Conference on Computer and Communications Security. New York: ACM, 2005: 340-353.

[18] Schwartz E J, Avgerinos T, Brumley D, et al. Q: Exploit hardening made easy[C]//Usenix Security Symposium. Berkeley: USENIX, 2011: 1-16.

[19] Federico A D, Cama A, Shoshitaishvili Y, et al. How the ELF ruined Christmas[C]//Usenix Security Symposium. Berkeley: USENIX, 2015: 643-658.

[20] 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):136-141.

[21] Hu H, Chua Z L, Adrian S, et al. Automatic generation of data-oriented exploits[C]//Usenix Security Symposium. Berkeley: USENIX, 2015: 177-192.

[22] Hu H, Shinde S, Adrian S, et al. Data-oriented programming: On the expressiveness of non-control data attacks[C]//IEEE Symposium on Security and Privacy. Los Alamitos: IEEE, 2016: 969-986.

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

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

[25] Shoshitaishvili Y, Wang R, Salls C, et al. SOK: (State of) the art of war: Offensive techniques in binary analysis[C]//IEEE Symposium on Security and Privacy. Los Alamitos:IEEE, 2016: 138-157.

[26] Reddi V J, Settle A, Connors D A, et al. PIN: A binary instrumentation tool for computer architecture research and educa-

tion[C]//Proceeding of the 2004 Workshop on Computer Architecture Education. New York: ACM, 2004, 22: 1-8.
[27] Hill J H, Feiock D C. Pin + +: An object-oriented framework for writing pintools[J]. Sigplan Notices, 2015,50(3):133-141.
[28] Determina. DynamoRIO dynamic instrumentation tool platform[CP/OL]. (2009-09-09)[2019-04-20]. DynamoRIO, 2009. http://dynamorio.org/.
[29] Nguyen A Q. Capstone: Next-gen disassembly framework. Blackhat USA[CP/OL]. (2014-08-14)[2019-04-20]. http://www.capstone-engine.org/BHUSA2014-capstone.pdf.
[30] Vivek Thampi. Udis86 Disassembler Library for x86/x86-64[CP/OL]. (2009-05-01)[2019-04-11]. http://udis86.sourceforge.net/.
[31] De Moura L M, Bjorner N. Z3: An efficient SMT solver[C]//Tools and Algorithms for Construction and Analysis of Systems. Berlin: Springer, 2008: 337-340.
[32] Chipounov V, Kuznetsov V, Candea G, et al. S2E: A platform for in-vivo multi-path analysis of software systems[J]. Architectural Support for Programming Languages and Operating Systems, 2011, 46(3): 265-278.
[33] Shoshitaishvili Y, Wang R, Hauser C, et al. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]//22nd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2015: 8-11.
[34] Stephens N, Grosen J, Salls C, et al. Driller: Augmenting fuzzing through selective symbolic execution[C]//23rd Annual Network and Distributed System Security Symposium. San Diego: The Internet Society, 2016:21-24.
[35] 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.
[36] Garmany B, Stoffel M, Gawlik R, et al. Towards automated generation of exploitation primitives for web browsers[C]//Proceedings of the 34th anual Annual Computer Security Applications Conference. New York: ACM, 2018: 300-312.
[37] Ruffell M, Hong J B, Kim H, et al. Towards automated exploit generation for embedded systems[C]//Information Security Applications—17th Internatinal Workshop. Berlin: Springer, 2016: 161-173.
[38] Luo L, Zeng Q, Cao C, et al. System service call-oriented symbolic execution of android framework with applications to vulnerability discovery and exploit generation[C]//International Conference on Mobile Systems, Applications, and Services. New York: ACM, 2017: 225-238.

Summary of research on software vulnerability auto exploit

SU Pu-rui[1,2], *HUANG Hua-feng*[1,2], *YU Yuan-ping*[1,2], *ZHANG Tao*[3]

(1. Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China;
2. School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100190, China;
3. China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: With the complexity of software increasing year by year, software security vulnerability has become one of the root factors of cyber-security threats. However, it is hard to meet the needs of vulnerability analysis and exploitation on labor. To analyze and exploit the vulnerabilities automatically in time, researchers have proposed several techniques, some of which can get good results. This paper presents a summary of the recent advances in four aspects, which include: control flow hijacking vulnerabilities automatic exploitation, heap-oriented vulnerabilities automatic analysis and exploitation, security mechanism automatic countermeasure, and the comprehensive framework of vulnerability automatic exploitation. Finally, we conclude the tendency of software vulnerability automatic exploitation to shed lights on potential future directions.

Key words: vulnerability; control flow hijacking; security mechanism; automatic exploitation

【责任编辑: 孙向荣】