文章编号:1007-1423(2020)14-0098-06 DOI:10.3969/j.issn.1007-1423.2020.14.021

基于Crash的漏洞利用自动生成系统

靳宪龙[1],徐世伟[2],黄雅娟[3]

(1.四川大学计算机学院,成都 610065; 2.中国人民解放军78156部队,兰州 730030;
3.国防科技大学国际关系学院,南京 210012)

摘要:
漏洞分析、自动化攻防对漏洞利用自动生成的需求越来越迫切,在分析现有方案的基础上,结合动态分析、混合符号执行等技术,提出基于Crash的漏洞利用自动生成方法C-Rex。该方法能够针对缓冲区溢出漏洞自动生成劫持控制流的漏洞利用样本。完成以下工作:①设计路径搜索算法复现崩溃路径并挖掘新崩溃点;②分析崩溃状态,对可利用性进行判定;③根据需求采用代码注入或代码复用技术生成劫持控制流的漏洞利用样本。通过对CTF题目及10款开源应用进行测试,C-Rex均能成功生成漏洞利用样本,证明该方法的有效性。

关键词:
网络安全; 二进制; 漏洞利用; 符号执行; 自动化

0 引言

在网络空间斗争日益激烈的形式下,安全漏洞始终是博弈双方关注的核心问题之一。安全漏洞的研究主要分为漏洞挖掘、漏洞分析及利用、漏洞评估、漏洞修复等。近年来,随着模糊测试、机器学习等技术在漏洞挖掘领域的应用,显著提升了漏洞挖掘的自动化水平。由于漏洞分析的复杂性及编写利用程序的灵活性,传统的漏洞分析与利用依赖具备专业知识的安全专家进行。随着漏洞挖掘自动化程度的提高,有限的人力资源已不足以有效应对漏洞的快速增长,为了能够实现对新披露漏洞的快速响应,进一步提升漏洞利用的自动化水平已迫在眉睫。

与此同时,美国国防部高等研究计划署(DARPA)通过研究得出,自动化攻防在当前技术体系下已可实现,并于2013年发起网络超级挑战赛(Cyber Grand Challenge,CGC)[1]。经过三年的预决赛,最终于2016的DEFCON CTF[2]中举办了自动化攻防领域中的首次人机对战,体现出自动化攻防对自动化漏洞利用的迫切需求与广阔前景。

漏洞自动利用是指通过给定的数据,如可执行程序、源代码、补丁程序等,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[3]。该过程非常复杂,包括准确定位漏洞点、通过符号执行技术生成能够触发漏洞的程序输入、获取运行时信息及崩溃现场数据、通过SMT对约束集合求解并最终生成漏洞利用样本。

自2008年D. Brumley首次提出基于补丁比对的漏洞利用自动生成方法APEG[4]以来,研究人员陆续提出AEG[5]、AXGEN[6]、MAYHEM[7]等方案。依据所需输入数据的不同,可将现有方案分为四类:基于可执行文件、基于源代码、基于补丁文件、基于异常输入的解决方案。本文通过分析CRAX[8]、FUZE[9]、Revery[10]、Poly-AEG[11]等基于异常输入的解决方案,发现基于异常输入的方案存在以下问题:①初始异常输入无法保证漏洞复现的成功率;②在成功复现漏洞的情况下,无法保证漏洞利用的成功率。

为了缓解以上问题,我们提出基于Crash的漏洞利用自动生成系统C-Rex。系统遵循以下设计原则:①采用混合符号执行技术复现崩溃路径;②复现失败时搜索相邻路径,尝试获取新崩溃点;③分析崩溃现场并判定其可利用性;④生成劫持控制流的漏洞利用样本。

1 方法概述

本文设计并实现了基于 Crash 的漏洞利用自动生成系统 C-Rex，系统采用动态分析、符号执行[12]等技术，结合异常输入对目标程序中所存在漏洞进行复现、判定其可利用性并自动生成实现控制流劫持的漏洞利用样本。系统架构如图 1 所示。

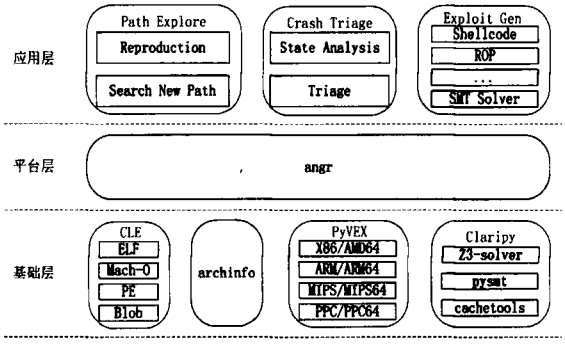

图 1 C-Rex 架构图

基础层中 CLE、archinfo 负责装载二进制对象及其所依赖的库并生成地址空间；PyVEX 将汇编代码转换为中间语言 VEX，以实现对不同架构的支持；Claripy 为符号求解引擎，完成变量符号化、约束生成、约束求解等。平台层实现了符号执行引擎及多种二进制程序分析方法。基础层与平台层共同构成了二进制分析平台 angr。基于该平台，C-Rex 实现了路径搜索、可利用性判定、利用样本生成。

路径搜索通过动态分析与混合符号执行技术复现程序崩溃，当复现失败时搜索临近路径并探索新崩溃点。由于真实环境及漏洞分析的复杂性，使得无法保证漏洞复现及生成漏洞利用样本的成功率。当复现失败时，虽然该崩溃路径无法利用，但一定程度上反应出该路径或周边路径在软件测试阶段并未被充分测试，通过对周边路径的深入探索，挖掘出新崩溃点的概率较高。因此，当复现失败时，路径搜索模块将采用动态符号执行技术，对崩溃点临近路径进行搜索，以获取新崩溃点。

可利用性判定通过收集崩溃现场的运行时信息，分析寄存器中的符号化状态、符号化指令指针、符号写等行为，对劫持程序控制流的行为进行监控，以实现对漏洞可利用性的判定。

漏洞利用生成负责生成多样性漏洞利用样本，利用方式可分为代码注入与代码复用两类。对于代码注入类利用，首先检查被用户输入数据控制的内存区域，搜索满足条件的连续内存区域以存放 Shellcode。对于代码复用类利用，主要采用 ret2libc[15]、ROP[16]技术，生成漏洞利用样本。

2 系统设计与实现

C-Rex 在执行过程中，首先加载目标程序及异常输入，结合具体执行与动态符号执行复现崩溃；复现失败时，重新探索程序状态空间，搜索临近路径，挖掘新崩溃点；其次，在获取到有效 Crash 的基础上，采集崩溃现场数据，分析 EIP、EBP 寄存器及内存中符号变量的布局，对 Crash 的可利用性作出判定；最后，选取利用方式并生成漏洞约束，对约束集合求解生成漏洞利用样本。执行流程如图 2 所示。

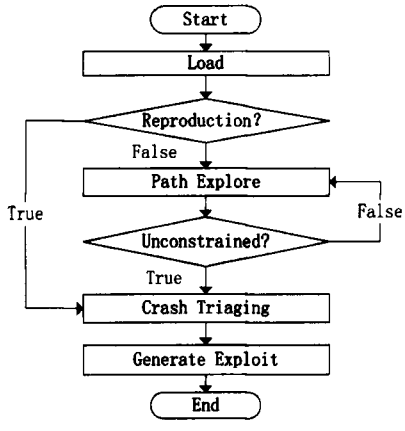

图 2 C-Rex 流程图

系统实现过程中，使用开源模拟器 QEMU[13]获取程序执行路径，通过该路径指引动态符号执行路径；使用 angr[12]作为符号执行引擎，实现 Crash 复现与新路径搜索；使用 Z3[14]作为约束求解器，以生成漏洞利用样本。系统由 Crash 路径搜索、可利用性判定、利用生成三个模块构成。

2.1 Crash 路径搜索

Crash 路径搜索模块负责复现崩溃并搜索临近崩溃点，由 Concolic Tracing、Path Explore 两部分构成。其中，Concolic Tracing 负责对崩溃进行复现，Path Explore 负责复现失败时对临近路径的搜索，如图 3 所示。

(1)Concolic Tracing

通过 QEMU 加载待分析的二进制程序及导致程序崩溃的异常输入，设置调试参数为“-nochain”、“-exec”、“-page”以获取程序执行路径，并将执行结果以日志的形式保存。设置符号执行引擎 angr 的路径搜索方式为“Tracer”、“Oppologist”。“Tracer”使得在采用动态符号

执行技术分析程序时，能够以 QEMU 日志中所记录的执行路径为导向，指引程序沿同一路径执行。“Oppologist”参数使得遇到不支持的指令时，将其具体化并使用 unicorn 引擎进行模拟，从而允许继续执行。随后，通过动态符号执行复现执行路径，若复现成功，将崩溃时的状态保存至 Crashed Stash 中；若复现失败，将程序执行结束前的最后两个状态保存至 Traced Stash 中，以供后续探索新崩溃点时使用。Stashes 为 angr 中的核心概念之一，用以对程序执行过程中的状态进行管理，通过 Stashes 可以根据需要对程序状态进行过滤、合并、移动等操作。

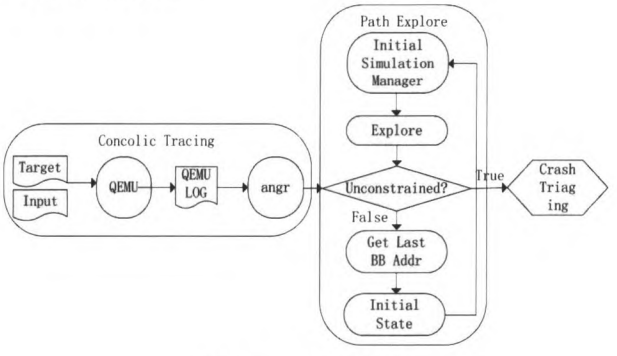

图3 Crash路径搜索

(2)Path Explore

当完成 Concolic Tracing 且未能成功复现崩溃时，C-Rex 执行 Path Explore 以探索临近路径，挖掘新崩溃点。首先，获取 Traced Stashes 中存储的 Basic Block 地址；随后，依据异常输入，初始化 initial_state 及 Simulation Manager；将获取到的 Basic Block 地址作为约束条件，依托 angr 进行动态符号执行，通过监测执行过程中是否出现 Unconstrained 状态，以判断是否成功挖掘出新崩溃点。

(3)参数优化

C-Rex 在实现过程中，通过优化以提高执行效率，首先对分析平台 angr 进行订制，通过冗余路径剪枝、状态合并等方式提高效率；其次通过增加前置约束，以缓解动态符号执行过程中出现的状态空间爆炸、路径选择等问题。优化内容如表 1 所示，Content 表示优化项目、Options 表示优化内容。

2.2 可利用性判定

应用程序发生崩溃时，通常情况下漏洞点与崩溃点不一致，崩溃往往滞后于实际发生错误的位置，这使得部分关键信息永久性缺失，进而无法生成漏洞利用样本。C-Rex 利用动态符号执行，获取程序运行时信息，依据内存、EIP、EBP 中符号变量的数量及可控程度，划分崩溃类型，如表 2 所示。“Crash Type”表示崩溃类型、“Definition”表示类型定义、“Exploitable”表示是否能够利用。

表 1 参数优化

Content	Options
State Options	MEMORY_SYMBOLIC_BYTES_MAP
	TRACK_ACTION_HISTORY
	CONCRETIZE_SYMBOLIC_WRITE_SIZES
	CONCRETIZE_SYMBOLIC_FILE_READ_SIZES
	TRACK_MEMORY_ACTIONS
State Plugin	SimSystemPosix
	SimStatePreconstrainer
Libc limits	libc.buf_symbolic_bytes = 3000
	libc.max_symbolic_strchr = 3000
	libc.max_str_len = 3000
	libc.max_buffer_size = 16384

表 2 崩溃类型

Crash Type	Definition	Exploitable
IP_OVERWRITE	Eip overwrite	True
PARTIAL_IP_OVERWRITE	Partial eip overwrite	True
UNCONTROLLED_IP_OVERWRITE	Uncontrolled eip overwrite	False
BP_OVERWRITE	Ebp overwrite	True
PARTIAL_BP_OVERWRITE	Partial ebp overwrite	True
WRITE_WHAT_WHERE	Write what where	True
WRITE_X_WHERE	Write x where	True
UNCONTROLLED_WRITE	Uncontrolled write	False
ARBITRARY_READ	Arbitrary read	False
NULL_DEREFERENCE	Null dereference	False
ARBITRARY_TRANSMIT	Arbitrary transmit	False
ARBITRARY_RECEIVE	Arbitrary receive	False

State 是 angr 中用以存储程序状态的对象，其中包含当前状态中的内存、寄存器、文件系统及任何会在执行中改变的数据。C-Rex 在判定崩溃的可利用性时，首先获取发生异常时的 state，通过 state.regs.ip、state.regs.bp、state.mem 访问此刻的寄存器及内存状态，判断其中符号变量的分布，针对 IP_OVERWRITE、BP_OVERWRITE、PARTIAL_IP_OVERWRITE、PARTIAL_BP_OVERWRITE、WRITE_WHAT_WHERE、WRITE_X_WHERE 六类异常，能够生成劫持控制流的漏洞利用样本。

2.3 利用生成

(1)代码注入

C-Rex 使用 shellcode 作为 Payload，将其注入具有执行权限且可控的内存空间，通过覆盖程序返回地址以劫持控制流，完成漏洞利用。在实现过程中，C-Rex 通过 state 中的 memory 接口获取崩溃时的内存布局，

查找其中连续且具有执行权限的地址空间，随后选取 shellcode 并计算其长度。在劫持程序控制流时，首先采取以 shellcode 起始地址覆盖函数返回地址的方式，通过函数返回地址、shellcode 地址、shellcode 长度构造约束条件并求解，若求解成功则生成以 shellcode 起始地址覆盖函数返回地址的漏洞利用样本。若求解失败，则搜索内存中存在的“jmp esp”指令，使用该指令的地址覆盖函数返回地址实现对程序控制流的劫持。通过已获取的“jmp esp”指令地址、崩溃时的内存布局、shellcode 长度，构造约束条件并求解，生成使用“jmp esp”作为跳板的漏洞利用样本。

(2)代码复用

代码复用攻击是指通过存在漏洞的二进制程序地址空间中已存在的代码片段构造恶意代码，主要包括：ret2libc、ROP、JOP 等技术。C-Rex 在实现过程中，主要针对 ret2libc 与 ROP 两种方式。

ret2libc 技术不以 Shellcode 为 Payload，而是将控制流导向库函数入口，通过执行库函数以实现恶意功能。由于库函数具有可执行权限，因此 ret2libc 可以绕过 DEP 的保护。在实现过程中程序依赖共享库 libc.so.6，其中包含大量可利用函数，在不考虑 ASLR 的情况下，共享库的加载基址不会改变，C-Rex 首先获取 system()函数、“/bin/sh”字符串在共享库中的偏移，结合基址计算出内存中的真实地址，劫持控制至 system()函数，通过执行 system(“/bin/sh”)获取本地 shell。

ret2libc 以函数为单位进行代码复用，在实际应用中缺乏灵活性，因此 C-Rex 实现了第二种复用方式 ROP(Return-Oriented Programming)。ROP 技术通过串接以 ret 指令为结尾的代码片段(gadgets)实现恶意功能。C-Rex 调用 angrop 查找所需 gadgets 并自动构建 ROP 链，最终实现 ROP 攻击。

3 实验与评估

为了验证 C-Rex 自动生成漏洞利用的效果，我们设计了两类实验。首先，通过存在漏洞的 CTF 题目，以验证 C-Rex 的有效性；其次，以 AEG 所使用的测试集为基础，从中选取 10 款包含漏洞的开源应用，评估生成效果。

(1)实验环境

本文中所使用的实验环境如下：处理器采用 Intel Core i7 2.5 GHz，内存为 DDR3 1600MHz 16GB，操作系统为 Ubuntu 18.04 x86_64，编译器为 GCC 7.3.0[17]，编译时使用-O2、-z execstack、-fno-stack-protector、-no-pie、-z norelro 选项，禁用 FORTIFY、NX、Canary、PIE、RELRO。

(2)有效性验证

使用 Insomni Hack CTF 2016[18]中的题目作为测试样例。存在漏洞的源码如图 4 所示，代码中为字符数组 component_name 分配的空间为 128 字节，为结构体 component 中的成员 name 分配的空间为 32 字节。溢出点位于第十行，将 cmp_name 中的数据复制至 cmp->name 时，未对缓冲区边界进行检查，导致缓冲区溢出进而对程序控制流进行劫持。

```
1.  char component_name[128] = {0};                    //缓冲区大小为 128 字节
2.  typedef struct component{char name[32]; ... }comp_t; //缓冲区大小为 32 字节
3.  comp_t *initialize_component(char *cmp_name) {
4.      int i = 0;
5.      comp_t *cmp;
6.      cmp = malloc(sizeof(struct component));
7.      ...
8.      printf("Copying component name...\n");
9.      while (*cmp_name)
10.         cmp->name[i++] = *cmp_name++;              //溢出点
11.     cmp->name[i] = '\0';
12.     return cmp;
13. }
```

图4 漏洞源码

使用 GCC 对源码进行编译，生成二进制程序 Demo_Vul，构造异常输入 36 个“A”。通过 C-Rex 加载 Demo_Vul 及异常输入，最终成功生成漏洞利用样本，劫持 Demo_Vul 控制流并获得本地 shell，验证了 C-Rex 的有效性。

(3)对比测试

以 AEG 的测试集为基础，从中选取 10 款开源应用，通过 C-Rex 以及 Rex 自动生成漏洞利用样本，表 4 总结了实验结果。其中，“Program”为开源应用名称、“Version”为所对应版本、“Input Src”为输入数据的类型、“Vul Type”为漏洞类型、“C-Rex”为 C-Rex 生成结果、“Rex”为 Rex 生成结果、“Advisory ID”为漏洞所对应 CVE/OSVDB 编号。

表 4 实验结果

Program	Version	Input Src	Vul Type	C-Rex	Rex	Advisory ID
Aeon	0.2a	Env.	Local Stack	Yes	No	CVE-2005-1019
Iwconfig	V.26	Arg.	Local Stack	Yes	No	CVE-2003-0947
nCompress	4.2.4	Arg.	Local Stack	Yes	No	CVE-2001-1413
CoreHTTP	0.5.3	Sockets	Remote Stack	Yes	No	CVE-2007-4060
Aspell	0.50.5	Stdin	Local Stack	Yes	No	CVE-2004-0548
Rsync	2.5.7	Env.	Local Stack	Yes	No	CVE-2004-2093
Xmail	1.21	Local File	Local Stack	Yes	No	CVE-2005-2943
xserver	0.1a	Sockets	Remote Stack	Yes	No	CVE-2007-3957
Htget	0.93	Arg.	Local Stack	Yes	No	CVE-2004-0852
glFTPd	1.24	Arg.	Local Stack	Yes	No	OSVDB-ID#16373

对比 C-Rex 与 Rex 的实验结果可以看出，由 Shellphish[19]开发的漏洞自动利用系统 Rex，其设计初衷在于参加 CGC 竞赛，因此对真实应用程序的支持并不完善，对于选取的 10 款开源应用均无法生成漏洞利用样本。本文所提出的漏洞利用自动生成系统 C-Rex 在实验当中，针对 10 款开源应用，均能够成功生成漏洞利用样本，有效验证了 C-Rex 的实用性。

4 相关工作

2008 年 D. Brumley 等人在 IEEE S&P 会议上，首次提出了基于补丁比对的漏洞利用自动生成方法 APEG。由于该方法操作性强，因此在实际测试过程中取得了良好的效果，并得到了安全研究者的普遍认可。其不足主要表现在：首先，对补丁程序的依赖；其次，随着软件复杂度的提升，简单的指令比对难以快速定位漏洞位置，与漏洞无关的补丁也对分析漏洞机理造成了一定的影响；再次，APEG 所生成的利用程序以 DoS 为主，对漏洞的利用程度有限。

2011 年 CMU 的 T. Avgerinos 等人在 NDSS 会议上首次提出了基于源码的漏洞利用自动生成方法 AEG。该方案能够针对栈溢出、格式化字符串漏洞生成具备控制流劫持能力的利用样本，是首个完整实现从漏洞挖掘到漏洞利用生成全流程的自动化解决方案。该方法的局限性主要体现在：首先，依赖源代码；其次，所针对的漏洞类型限于栈溢出与格式化字符串，无法处理整型溢出、堆溢出等类型的漏洞；最后，生成的利用程序对系统环境依赖严重。

2012 年 Shih-Kun Huang 等人提出了基于异常输入的漏洞利用自动生成系统 CRAX。相较于 AEG 采用动态分析技术获取崩溃点的动态信息，CRAX 采用回溯分析获取漏洞点的动态信息，提高了所生成的漏洞利用程序的成功率。该方法的局限性主要体现在，对导致程序崩溃的异常输入的强依赖，能否成功生成利用程序与异常输入数据密切相关。

2018 年 Wei Wu 等人在 USENIX 安全会议上首次提出了基于内核 UAF 漏洞的漏洞利用自动生成系统 FUZE。该方案综合运用模糊测试、符号执行、动态追踪等技术，针对内核 UAF 漏洞自动生成利用程序。通过对 15 个内核 UAF 漏洞进行测试，FUZE 成功生成利用程序并一定程度上绕过保护机制，证明了该方法的有效性。

2018 年 Yan Wang 等人在 CCS 安全会议上提出了针对堆漏洞的解决方案 Revery。该方案在实现过程中，首先，分析漏洞位置和相关内存布局，建立异常对象内存布局图、内存布局贡献者图；其次，以异常对象的内存布局为导向，采用定向 Fuzzing 探索替代路径，并结合污点分析技术，在替代路径中探索可利用状态；最后，确定拼接点、拼接路径，组合生成漏洞利用路径。求解约束集合并生成漏洞利用样本。在测试过程中，Revery 以 19 个 CTF 赛题为样本，对其中 9 个程序自动生成漏洞利用，5 个可以自动化转化为可利用状态，证明了该方法的有效性。

5 结语

本文提出基于 Crash 的漏洞利用自动生成方法 C-Rex，该方法包含 Crash 路径搜索、可利用性判定、利用生成三个环节。在实验中，能够对 CTF 题目及 10 款开源应用成功生成劫持控制流的漏洞利用样本，证明了 C-Rex 的有效性。C-Rex 采用 angr 作为符号执行引擎，但目前 angr 缺少必要的系统调用支持，使得 C-Rex 对大型应用的支持并不完善，同时 C-Rex 所能解决的漏洞限于经典的栈溢出，并不适用于更复杂的漏洞类型，以上问题的解决有待于更进一步的研究。

参考文献：

[1]DAPRA. DARPA Cyber Grand Challenge. https://github.com/CyberGrandChallenge.

[2]DEFCON. https://www.defcon.org.

[3]SUN Hong-yu, HE Yuan, Wang Jice, et al. Application of Artificial Inteligence Technology in Security Vulnerabilities[J]. Journal of Communications, 2018, 39(8):1-17.

[4]Brumley D., Poosankam P., Song D., et al. Automatic Patch-Based Exploit Generation is Possible:Techniques and Implications[C]//2008 IEEE Symposium on Security and Privacy(sp 2008). IEEE, 2008.

[5]Avgerinos T., Sang K.C., Rebert A., et al. Automatic Exploit Generation[J]. Communications of the ACM, 2014, 57(2):74-84.

[6]Heelan S. Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities[D]. University of Oxford,2009.
[7]Cha S.K.,Avgerinos T.,Rebert A.,et al. Unleashing Mayhem on Binary Code[J],2012.
[8]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. IEEE,2012.
[9]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),2018:781-797.
[10]WANG Y.,ZHANG C.,XIANG X.,et al. Revery:From Proof-of-Concept to Exploitable[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM,2018:1914-1927.
[11]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. Springer International Publishing,2013.
[12]Shoshitaishvili Y.,Kruegel C.,Vigna G.,et al. SOK:(State of)The Art of War:Offensive Techniques in Binary Analysis[C]//2016 IEEE Symposium on Security and Privacy(SP). IEEE Computer Society,2016.
[13]QEMU. https://www.qemu.org.
[14]Z3. https://github.com/Z3Prover/z3.
[15]Designer S. Getting Around Non-Executable Stack(and Fix)[J]. http://ouah.bsdjeunz.org/solarretlibc.html,1997.
[16]Shacham H. The Geometry of Innocent Flesh on the Bone:Return-Into-Libc Without Function Calls(on the x86)[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security,2007:552-561.
[17]GCC. The GNU Compiler Collection. http://gcc.gnu.org.
[18]Insomni Hack CTF. https://insomnihack.ch.
[19]Brooks T.N. Survey of Automated Vulnerability Detection and Exploit Generation Techniques in Cyber Reasoning Systems[J],2017.

作者简介:
靳宪龙(1988-),男,山东章丘人,硕士研究生,研究方向为二进制安全、漏洞挖掘
徐世伟(1985-),男,甘肃兰州人,本科,研究方向为信息通信
黄雅娟(1987-),女,甘肃天水人,硕士研究生,研究方向为基础理论
收稿日期:2020-02-18　　修稿日期:2020-02-26

Crash-based Automatic Exploit Generation System

JIN Xian-long[1], XU Shi-wei[2], HUANG Ya-juan[3]

(1. College of Computer Science, Sichuan University, Chengdu 610065; 2. 78156 Troop, PLA, Lanzhou 730030;
3. Graduate School of National University of Defense Technology, Nanjing 210012)

Abstract:
Vulnerability analysis, automated attack and defense are increasingly urgent for the automatic generation of exploits. Based on the analysis of existing solutions, combined with dynamic analysis, dynamic symbolic execution and other technologies, a Crash-based automatic exploitation method for C-Rex is proposed. This method can automatically generate exploits that hijack control flow for buffer overflow vulnerabilities. Mainly completes the following tasks: ①designing a path search algorithm to reproduce the crash path and mining new crash points; ②analyzing the crash status and triaging the crash; ③Use code injection or code reuse techniques to generate exploits that hijack control flow as needed. By testing CTF topics and 10 open source applications, C-Rex was able to successfully generate exploits, proving the effectiveness of the method.
Keywords:
Network Security; Binary; Exploit; Symbolic Execution; Automation