计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200600152

反向调试技术研究综述

徐建波　舒　辉　康　绯

信息工程大学数学工程与先进计算国家重点实验室　郑州 450001

(xujianbo2018@163.com)

摘　要　在软件的开发测试部署过程中,调试工作耗费了开发人员非常多的精力和时间,有时一个很难被发现的错误会导致多次重启调试。反向调试是软件调试的一种技术,无需重启即可向后查看运行的指令及状态,这能够大大提高软件调试的速度,降低软件开发的难度,有效修复程序运行时发生的错误。该技术的核心问题是运行状态的恢复,目前针对该问题的解决方法主要有状态保存和状态重构。文中主要从反向调试的原理、学术研究、产品实现、技术应用等方面梳理其发展情况,对该技术进行分析研究,总结了基于时间和基于指令的状态保存反向调试技术以及两种反向执行重构状态的方法,并提出了有关记录重放程序执行、定位分析软件错误、反向数据流恢复这3方面的应用,可为反向调试技术的研究应用提供一定的参考。

关键词:调试;反向调试;状态保存;状态重构;反向执行;软件错误

中图法分类号　TP311

Summary on Reverse Debugging Technology

XU Jian-bo,SHU Hui and KANG Fei

State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract　In the process of software development and test deployment,debugging consumes a lot of developers' energy and time. Sometimes,in order to find a critical bug,debugging needs to restart many times. Reverse debugging is a technology of software debugging. It can check running instructions and status backward without restarting,which can greatly improve the speed of software debugging,reduce the difficulty of software development,and effectively repair the errors during the running of the program. The core issue of this technology is the recovery of running state. At present,current solutions are state preservation and state reconstruction. This paper mainly reviews the development of reverse debugging from the aspects of principle,academic research,products implementation and technical application. It focuses on the time-based and instruction-based state preservation and two methods of reverse execution reconstruction states. It summarizes 3 specific applications of record replay program execution,location analysis software error,reverse data flow recovery,which provides a reference for the research and application of reverse debugging technology.

Keywords　Debugging,Reverse debugging,State save,State reconstruct,Reverse execution,Software failure

1　引言

在计算机领域,调试(debug)主要是软件调试,指重现软件错误、分析定位错误原因、解决软件问题的过程[1]。其中,软件错误产生的原因主要有开发人员逻辑不全面、设计不完整、编码失误[2]等。传统的调试技术方法主要有3种:1)程序设计时开发人员使用打印、断言语句和测试套件发现编写程序的错误;2)利用专业的检测分析工具发现软件的内存访问、未分配使用等特定类型的错误;3)使用调试器在软件运行时设置断点、单步执行及变量监视等查看软件运行情况。在软件的调试运行中,根据指令执行的方向可以将调试分为正向调试和反向调试。

反向调试相比正向调试,能有效减少正向反复调试所耗费的时间代价和降低人工成本,提高调试效率。此外,复杂软件在开发过程中有时会出现难以复现的偶发性软件问题。针对此问题,直接从错误点反向调试,回溯执行状态,对于排除此类错误具有十分重要的作用。近年来,反向调试技术在学术领域的研究和发展主要有状态保存和状态重构两个方向,在软件错误分析定位[3]中取得了较好的效果。在实际应用方面,反向调试功能开始被部署在主流调试器(如 GDB,Win-

到稿日期:2020-06-24　返修日期:2020-08-24　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。

基金项目:国家重点研发计划(2016YFB08011601)

This work was supported by the National Key R&D Program of China(2016YFB08011601).

通信作者:舒辉(415314938@qq.com)

Dbg Preview)中，在软件开发测试部署调试过程中发挥了越来越重要的作用。

本文第 2 节介绍了反向调试技术的原理；第 3 节研究分析了 4 种反向调试技术；第 4 节介绍反向调试技术的 3 种实际应用；最后通过对具有代表性的反向调试器的测试，总结展望反向调试技术的发展。

2 反向调试技术的原理

2.1 反向调试的定义

反向调试指调试过程中在关键位置植入断点，程序在向前执行时，能够回退执行，到达先前执行过的断点[4]，查看先前的程序执行状态。这些程序执行状态包括通用寄存器和内存中的变量值、堆栈数据、当前程序位置和状态寄存器的值等。反向调试技术可以通过多种技术实现，如记录重放、跟踪、确定性重新执行、反向执行及检查点(或快照)等。

2.2 循环调试与反向调试

循环调试是调试的主要形式，在程序(代码)中插入断点，程序运行时一次执行一条或多条指令，遇到断点后暂停，就能实时查看程序执行过程中的状态。与循环调试不同，反向调试可恢复程序之前执行的状态，反向到达执行的断点。程序正常的执行路径如图 1 所示，经过顺序执行、条件执行到达程序的出口。

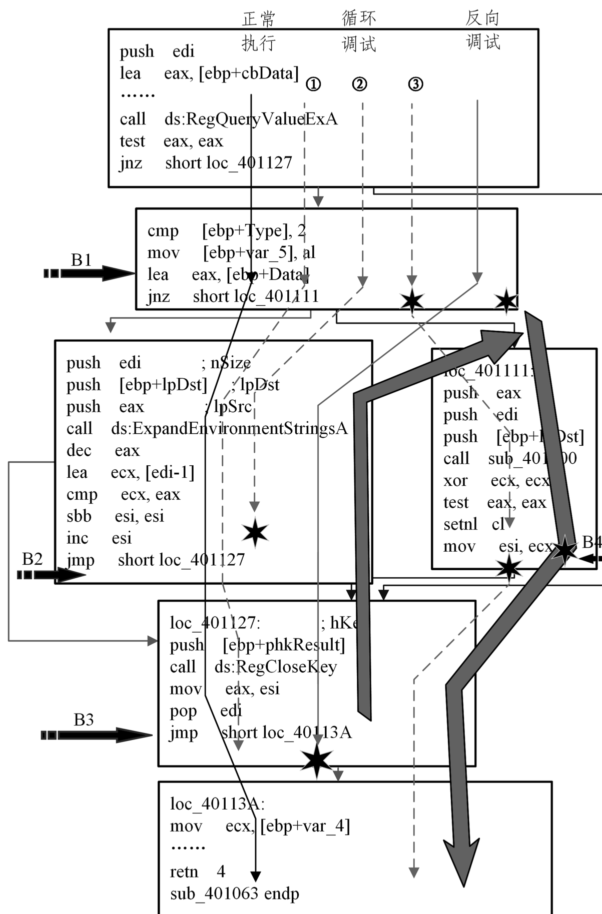


图 1 循环调试、反向调试流程

Fig. 1 Flow chart of cycle debugging and reverse debugging

假定调试过程需要执行到断点 B4 查看程序状态，循环调试和反向调试具有不同的调试过程。循环调试其在第 1 次执行时只能停在 B3，设置断点 B2，然后需要重启程序，执行到 B2，设置断点 B1，重新执行到 B1 后，修改判断语句的值，才能执行到 B4。对于反向调试，程序停在 B3 后，不需重启程序，直接可以回退到断点 B1，修改状态寄存器的值，即可正向执行到 B4。

如图 1 所示，反向调试可简单地理解为，不经过程序重启，从断点 B3 的状态到达断点 B1 后能够查看程序执行状态。其实现的思路主要有两种：1)状态保存，即在程序正常执行时，保存执行过程中的程序状态。当需要回退时，首先确定断点 B1 的位置，然后恢复到断点 B1 对应的程序执行状态。2)状态重构，即从断点 B3 处指令执行的状态，反向执行指令，重新构建 B3 到 B1 中间程序状态，最终得到断点 B1 处的程序执行状态。状态保存反向调试主要有基于时间保存状态(Undo Debugger，UndoDB)、基于指令记录状态(GNU Debugger，GDB)两种。状态重构反向调试主要是基于反向代码执行生成状态和基于硬件记录执行指令重构执行状态。

2.3 反向调试需解决的问题

实际的程序从断点 B3 到达 B1 的反向调试运行可能经历多个跳转、多重调用等非常复杂的情况，上述两种实现方法中均存在一些问题。1)内存开销大。基于状态保存的反向调试技术，在程序执行期间需要保存额外的执行状态才能向后回溯，但是软件的状态保存，特别是大型软件的状态保存，需要大量的内存空间，因此对调试工作带来很大的影响。2)状态记录增加时间。在程序执行期间的状态写入和读取需要耗费额外的时间，增加总体调试时间。3)多处理器多线程并行问题。目前应用软件以多核多线程为主，运行过程的中间结果具有不确定性，对于多线程并行问题的研究集中在记录和确定性重放调试上[5]，但还没有很好的解决方案。4)过程输入带来不确定性。针对程序运行过程中存在用户输入、文件数据读取等改变运行时的状态的情况，多数反向调试不能进行实时处理。5)不可逆的程序执行。在一些通过反向执行重构状态的反向调试中，部分指令没有逆变换，甚至破坏了其他执行状态，因此先前状态无法恢复。6)其他问题。如用户级和内核级指令切换、本机调试和分布式系统跨平台调试、状态保存的粒度等。

3 反向调试技术的研究发展

3.1 总体发展情况

从 1992 年第一款具有反向执行功能的调试器 Turbo Debugger 到 2020 年的 CauDEr，反向调试领域涌现出针对不同的调试级别(如用户级、系统级)、单线程多线程处理、不同的操作系统及不同调试对象的产品和技术的研究，如表 1 所列。

表 1 反向调试的研究及产品

Table 1 Reverse debugging research and products

时间	名称	出处	系统	调试对象	描述
1992	Turbo Debugger	Borland	Windows	汇编代码	基于单步执行汇编代码并建立历史记录[6]，可反向执行到记录中的备份指令，返回至程序可能存在错误的位置。
2003	MULTI	Green Hills	Linux	嵌入式设备	MULTI 调试器中的 Time Machine 功能，实现了基于调试探针和“JTAG”调试接口在硬件中跟踪程序执行，回溯基于时间的、仅用于单个处理器的系统级的跨目标调试
2005	Simics	Virtutech	Linux Windows	二进制程序	基于完整系统的模拟器，具有全系统反向调试和反向执行的功能[7]。该模拟器可以处理跨目标、多处理器、多操作系统的反向调试操作，可恢复执行中的任何时间点
2006	UndoDB	Undo Software	Linux	源程序	Linux 的双向调试器，根据“模拟纳秒”设置检查点[8]，可执行用户级别的反向调试，支持执行反向断点，可处理多线程程序
2009	GDB7.0	GDB	Linux	二进制程序	基于指令记录程序运行状态，可重放执行历史[9]，有丰富的反向调试指令，可执行用户级、主线程调试
2009	Visual Studio	Microsoft	Windows	部分源程序	基于记录重放调试[10-11]，IntelliTrace 通过时间点记录特定事件，可执行用户级、单线程及基于主机的操作
2015	RR	Mozilla	Linux	源程序	可进行单进程用户级多线程调试，基于状态保存、记录及重放确定性程序执行[12]
2016	POMP	学术研究	Linux	二进制程序	基于硬件记录执行指令[13]，反向执行以构建执行状态，通过控制流分析和污点分析调试程序
2017	WinDbg Preview	Microsoft	Windows	二进制程序	基于 WinDbg 时间旅行调试 TTD 的记录重放[14-15]，可以支持用户级程序，支持并行多线程
2018	REPT	Microsoft	Windows	二进制程序	基于 Intel 处理器跟踪 PT 技术[16]，从核心转储文件中反向执行以构建执行状态，恢复数据流
2019	RENN	学术研究	Linux	二进制程序	基于反向执行的状态重构，利用递归神经网络（Recursive Neural Network，RNN）来学习与内存访问有关的二进制代码模式[17]，提高内存别名的解析度，防止调试软件崩溃
2020	CauDEr	学术研究	Linux	源程序	基于执行指令的状态重构，针对并发程序的反向调试[18-21]，反向执行旨在撤销正向执行指令的效果，需要保存日志

3.2 基于时间的状态保存

基于时间保存状态的反向调试是在系统运行或调试器运行过程中，根据时间间隔对程序执行过程中的状态进行保存，在后续的反向调试中恢复到先前某一时刻的运行状态，主要的调试器有 MULTI，UndoDB，Visual Studio 等。

UndoDB 是 Linux 系统上针对 C/C++的交互式可逆调试器[22]，在调试中能够快速定位错误。该调试器提供了一个时间基准，通过“模拟纳秒”的计数来识别程序运行中的任何点，利用时间和非确定性影响构建“事件日志”。

UndoDB 最重要的是日志和快照[23]。日志基于时间保存非确定性事件，保证程序反向调试时能够进行确定性执行，在事件日志中存储的不确定性来源主要有：1）所有系统调用；2）共享内存或设备映射读写；3）异步信号；4）线程切换和线程交互；5）非确定性机器指令。快照（也称检查点）指程序执行特定时间间隔对程序的地址空间进行存储，使调试器能够恢复执行所需的内存和寄存器的任何位置。

在反向调试重放执行时，从程序的当前状态选定要恢复的快照（先前时间点），从该快照处重放程序的执行过程，这样程序状态就像是重新执行到先前执行的位置。程序从快照处正向执行时，通过事件日志替换不确定事件，能够确保到达下一个快照，按确定的路径执行，如图 2 所示。

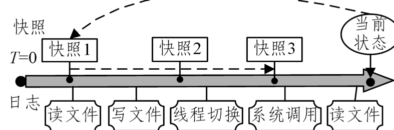


图 2 UndoDB 反向调试的过程

Fig. 2 UndoDB reverse debugging process

UndoDB 是基于时间保存状态进行反向调试技术的典型应用，直观反映了此类技术的优缺点。其优点是通过记录非确定性事件，调整快照保存的时间间隔，使反向调试的时间和内存消耗能够达到较满意的效果；支持并发线程，通过全局锁和线程抢占将线程的执行序列化，实现确定的记录重播。其缺点是一次只能调试一个进程，不支持多个进程之间的同步调试；只记录用户空间内存，不支持内核调试，不能处理系统调用。

3.3 基于指令记录的状态保存

基于指令记录保存状态的反向调试与基于时间的状态保存方法相似，只是记录事件日志的方式不同。基于指令记录的状态保存，主要是对执行的指令及状态改变进行记录，通过读取日志文件恢复执行状态，比较典型的例子是 2009 年推出的 GDB7.0[24]。GDB7.0 之后的版本引入了一个新的模块 record target，提供反向调试功能[25]，可以将选定的执行过程的代码及状态保存下来，为用户提供反向调试的一些基本命令[26]，如 reverse-continue，reverse-next，reverse-finish 等。

GDB 基于指令保存执行状态，其指令主要有源代码和汇编两种，基于性能考虑，限制了最大保存指令的数量。在反向调试程序的过程中（见图 3），有记录（record）和重放（replay）两种模式[27]，其中，记录主要是在程序向前执行时将执行的指令及状态保存到日志中，记录指令在运行时对状态的改变，即每条指令对内存、堆栈、寄存器及外设等的操作；重放主要是在程序向后执行时读取日志，恢复到先前指令执行时的位置及状态，但实际反向执行过程中没有执行指令。反向执行可以是单步的、多步的及断点执行的，甚至可以是支持在程序代码执行过程中位置新设断点，然后恢复到该位置处查看程序状态。

```
kali@kali:~/test-code$ gdb -q a.out
Reading symbols from a.out...
……
……
(gdb) start
(gdb) record
(gdb) c
Breakpoint 2, main () at threads-rd.c:23
23          if (sw)
(gdb) c
create id:0 thread id:0, p:0x4055b0
……………………
create id:5 thread id:5, p:0x4058d0
program runned time:4.516137s
Thread 1 "a.out" hit Breakpoint 3, main () at threads-rd.c:42
42          pthread_exit(NULL);
(gdb) rc
Thread 1 "a.out" hit Breakpoint 2, main () at threads-rd.c:23
23          if (sw)
(gdb) rn
21          int sw=1;
(gdb) n
Thread 1 "a.out" hit Breakpoint 2, main () at threads-rd.c:23
23          if (sw)
(gdb) set sw=0
GDB is in replay mode,…… Write memory at address 0xbffff294?(y or n) y
(gdb) c
passed the function!
program runned time:8.365730s
Thread 1 "a.out" hit Breakpoint 3, main () at threads-rd.c:42
42          pthread_exit(NULL);
(gdb) info record
Active record target: record-full
Record mode:
Lowest recorded instruction number is 1.
Highest recorded instruction number is 79518.
Log contains 79518 instructions.
Max logged instructions is 200000.
```

图 3 GDB 反向调试示意图

Fig. 3 Example of GDB reverse debugging

GDB 反向调试只能在单个目标线程中记录每个目标指令的效果，记录速度非常慢，占用较大的存储空间，并且无法根据时间进行调试。

由 Mozilla 团队开发的 RR[28-30] 是建立在 Linux 系统基础上的 C/C++调试工具，其反向调试性能比 GDB 好，通过记录一次软件运行，就可以进行多次确定性的重放调试。该工具主要有以下特点：1）低开销，特别是对于单线程程序；2）可进行用户级程序的记录重放；3）可记录重放调试包括容器在内的多个进程；4）可在多台机器之间移植调试。RR 调试的是确定性跟踪记录文件，只是对确定性执行的地址空间、寄存器内容及系统调用数据的重放运行，但不是实时的非确定性执行。该调试工具只针对单核计算机，强制应用程序的线程在单个内核上执行，对于多核并行的应用程序，执行速度和记录速度下降较大。RR 基于 X86 处理器，对系统调用支持不够完善。

3.4 基于反向代码执行的状态重构

基于状态重构的反向调试，主要是从程序运行的当前状态，通过反向执行代码指令，生成之前的程序状态。其状态重构的方法主要有基于反向代码的反向执行和基于执行指令的反向执行。

基于反向代码执行调试，在获得反向代码的情况下，通过反向代码的执行到达需要恢复状态的位置，完成状态的回溯。例如，$X=X+1$ 的反向代码是 $X=X-1$，若正向执行后 X 的值为 5，则通过反向执行后可得到执行前的值为 4。反向代码生成主要有静态反向代码生成、基于动态切片的反向代码生成及动态反向代码生成 3 种方法。由文献[31-33]的工作可知，静态反向代码生成使用路径敏感的静态分析对汇编程序预先生成反向代码，再基于基本块分析汇编指令，生成反向指令，添加必要的指令状态保存，构建反向程序。基于动态切片的反向代码生成[33-34]，在程序执行时通过动态切片剔除不相关指令，利用修正值图（Modified Value Graph，MVG）生成反向代码。Yi[35] 主要建立了一个框架，在运行调试器调试程序的过程中即时生成反向代码，运行时自动计算反向代码片段，记录指向程序位置的指针，并根据变量的当前值和先前运行的语句找出变量的先前值。

反向执行重构程序状态，主要利用 3 种技术的组合：重新定义技术、使用后提取技术以及状态保存技术。1）在反向执行期间，通过重新执行原始定义来重新计算该值；2）在反向执行过程中从以前的使用中提取该值；3）在正向执行期间保存该值，然后在反向执行期间恢复该值。恢复值可能需要恢复其他值，例如，要使用另一个也被覆盖的寄存器的值来重新生成一个寄存器的值。重新定义和使用后提取技术通常以递归方式进行应用，其中，依赖项中的值进行顺序恢复。

通过反向代码生成来构建真正的反向执行，从而进行程序调试，可以减少对执行过程状态的保存，开销较少。但是，程序代码的可逆求解及程序执行的不确定性一直是一个比较大的问题，这导致反向代码的生成比较困难，在反向执行调试中，可能回不到之前的执行状态。

3.5 基于执行指令记录反向执行的状态重构

基于执行指令记录反向执行的状态重构，主要指在程序执行过程中，利用软硬件技术先记录执行过的指令和程序的最终状态，再通过这些指令的反向执行恢复执行历史，以查看先前的状态。执行状态不是通过保存而是通过重构获得的，其与反向代码执行的区别在于状态的恢复不需生成反向指令。基于执行指令反向执行的状态重构需要执行指令的记录和反向执行所需的起始状态数据。其中，执行指令的记录大多基于 Intel PT 技术[36] 和插桩技术，起始状态数据主要是指 dump 文件和程序执行停止时的状态。

Xu 等[37] 设计了 POMP 工具，该工具主要用于 32 位的 Linux 系统（后续改进用于 64 位系统的 POMP++[38]），其反向调试的目标是帮助开发人员查找软件错误，特别是内存损坏漏洞。反向调试过程主要是从程序崩溃状态反向追溯到程序崩溃点，找到导致程序崩溃的主要原因，如运行非法指令和解引用非法地址。POMP 实现反向调试定位崩溃的步骤主

要有3步：1)重构指令序列，通过Intel PT在程序运行的过程中记录运行的指令，即从程序运行到崩溃这一过程的指令序列；2)反向执行状态恢复，在反向执行过程中通过假设检验来解决内存别名问题，恢复软件崩溃前的反向执行时的执行状态；3)通过反向污点分析定位崩溃原因，对于运行非法指令和解引用非法地址的情况，POMP分别指定了EIP和含有非法地址的寄存器作为sink的方法来定位造成程序崩溃的这一组指令。该方法假定崩溃的原因包含在执行的指令内，但崩溃后保存的信息可能无法反向执行以完全恢复之前的状态。

Cui等[39]设计了新的反向执行机制，对代码进行反向分析，结合核心转储中的信息进行反向调试，重建程序崩溃之前程序的状态，主要工具有Retracer[39]，REPT[16]，Kernel REPT[40]等。其中，代表性的REPT技术的核心是使用执行历史记录和程序当前状态重构来生成程序执行的先前状态。其首先是在线硬件跟踪，即使用Intel PT技术的多程序循环缓冲区模式来跟踪进程中所有线程的用户空间执行，以支持配置循环缓冲区大小，最终状态和PT跟踪记录将被保存在内存转储中；然后进行离线二进制分析，以PT跟踪的内存转储文件为输入，通过解析跟踪文件来重建控制流，将本地的指令转换成中间表示IR的特殊操作码和操作数，利用纠错机制来检测和解决恢复状态存在的冲突，向前和向后推导分析直到收敛，输出为每个线程的恢复的执行历史。REPT基于英特尔PT技术收集执行指令，能够处理多核并发和多线程二进制程序，其运行开销较少。该系统目前只支持Windows系统用户级的程序反向调试，不能用于系统级的调试，且高度依赖核心转储的完整性，因此当发生内存损坏时，可能无法反向执行或反向调试。通过反向执行重构的运行状态与实际执行历史有一定的差异，对于I/O密集型软件，该系统难以解决各种外部输入导致的不确定性问题。

3.6 测试评价

基于状态保存的反向调试相对容易实现，且具有成熟的产品。其明显的缺点是，需要记录较多的执行状态，对状态的读取和恢复需要消耗大量的内存和时间。通过调整状态保存间隔和非确定性事件记录可以优化反向调试的速度，但对于多核多线程的程序调试，其主要是将它们转换为单核单线程，这与实际程序运行存在一定差异。基于状态重构的反向调试通过程序代码的反向来执行重构状态，对于执行过程中的状态依赖较少，其反向调试的额外开销主要来自反向代码生成、执行指令记录和最终执行状态等，在查找定位崩溃点的应用中有较好的效果。但是其还存在重构的执行状态只是接近真正的执行状态、不可逆指令、不可逆状态和内存损坏不可恢复等问题。为了解和对比反向调试的性能，本文对3个应用相对广泛的反向调试工具进行了测试。测试程序是用户级多线程，具有源代码和跳转分支语句。表2中反向调试工具下载安装时间为2020.06，测试系统为Linux Kali 5.5 i686(VMware)，Windows 10.0.18362。原始运行时间为源程序编译后在调试器中运行的时间。通过对比可知，Linux系统下RR反向调试的时间和存储消耗都优于GDB，但是只有GDB能够通过改变状态来改变执行流程。只有Windows系统下的WinDbg Preview能够较好地支持多线程调试。

表2 典型反向调试工具的测试

Table 2 Typical reverse debugging tool tests

工具	原始运行时间/s	状态保存运行/s	时间比	编译后文件/kB	额外存储/kB	存储比	多线程处理	改变执行流程	总体评价
GDB	0.001496	4.691598	3136	18.3	51200	2797	支持主线程调试，不支持其他线程	是	速度较慢，功能完善，能够改变执行流程
RR	0.001496	0.002634	1.76	18.3	362.5	19.8	强制单核单线程，程序重放执行状态确定	否	反向调试速度优于GDB，不能调试线程
WinDbg Preview	0.8098	0.8476	1.05	37	24567	663	可以处理并行多线程	否	重播速度快，但不能更改执行轨迹，存储消耗增加较大

4 反向调试技术的应用

4.1 记录重放程序执行

基于状态保存的反向调试可在程序执行过程中记录并保存程序状态，对这些程序状态的回退重放便于对程序运行过程进行分析，能够更快定位关注的程序状态和位置，找到对其施加影响的指令。通过重放可以对程序运行的实际过程与设计设想的运行过程进行对比，能够清楚地看到程序做了什么，是否按照编程预期执行。在程序开发测试过程中，通过记录重放运行，增强对程序代码的分析能力，加快开发项目的速度，优化编程，减少调试时间，节约成本。在UndoDB的实际应用中，开发人员能记录、反向执行和重放程序代码，减少了66%的调试时间[8]，工作效率更高。Mozilla人员开发的RR反向调试工具一次记录程序执行可以多次重放运行，因此，如果记录包含了程序执行错误的情况，可以通过回放来进行定位。

4.2 定位分析软件错误

查找和修复间歇性错误：软件间歇性错误在特定的条件下才会发生，具有随机性，常规调试难以捕捉，但是只要间歇性错误在反向调试中出现过一次，随机条件就会被记录到软件执行历史中，开发人员反向执行就必定能追溯到错误位置。离线调试错误：用户在使用软件中出现的错误时，开发人员即使不在现场，也可以直接通过状态保存或状态重构恢复执行历史，并在其他主机上进行调试分析。POMP通过指令记录的反向执行和反向污点分析可以找到导致程序崩溃的最小指令集。REPT通过反向执行状态重构，对16个软件错误进行反向调试分析，其中14个都能被正常诊断。

4.3 反向数据流恢复

程序分析过程中可能需要分析内存数据(如shellcode、木马释放前的二进制块)，利用反向执行和反向调试技术进行反向数据流分析，可以定位到这些数据在程序执行输入时在文

件中的具体位置,甚至可以通过执行路径反向执行来分析数据块在程序执行中的变换关系,定义约束和求解,替换部分原始数据输入。GDB反向调试能够保存执行的指令和数据,反向调试可以撤销指令对数据的影响,通过标记监视分析重要数据可以回溯数据的来源。REPT根据硬件记录和部分指令的正反向执行迭代,可以利用程序的最后状态和常量值,找到控制依赖关系,恢复执行过程中的数据。利用Akgul等[33]和Lee等[34]的研究可以生成反向代码,因为正向代码执行的输入即是反向代码执行的输出,所以可以通过标记反向代码执行的输入,定位分析正向代码执行的输入。

结束语 反向调试技术能够显著提高软件调试效率,可以加快对软件中复杂错误的搜索速度,有望能够大规模应用于软件调试。相应的内存开销问题[41]、多线程并行处理问题[42]和实时交互问题也需要更深入的研究。随着硬件虚拟化技术[43]、软件分析技术和人工智能的不断发展,将这些技术应用到反向调试中可以更好地解决上述问题。反向调试技术除了可以被广泛地应用于各类软件的正向开发以外,还可以在网络空间安全领域,针对软件安全的漏洞分析、代码逆向和行为分析等方面发挥重要作用。

参考文献

[1] ZHANG Y K. Software Debugging [M]. Chongqing: Publishing House of Electronics Industry. 2008: 3-5.

[2] LIU M L, YANG X S, ZHAO L, et al. Discrete Characteristic-based Test Execution Selection for Software Fault Localization and Understanding[J]. Computer Science, 2016, 43(3): 179-187.

[3] CAO H L, JIANG S J, JU X L. Survey of Software Fault Localization[J]. Computer Science, 2014, 41(2): 1-6, 14.

[4] ENGBLOM J. A review of reverse debugging[C]// Proceedings of the 2012 System, Software, SoC and Silicon Debug Conference. IEEE, 2012: 1-6.

[5] JIANG S, WANG W W, JIANG L, et al. A Fast bug-locating record and replay debugging system[J]. Computer Applications and Software, 2016(10): 219-222.

[6] ENGBLOM J. Reverse Debugging Products[EB/OL]. [2020-5-12]. http://jakob.engbloms.se/archives/1564.

[7] ENGBLOM J, ARNO D, WERNER B. Full-System Simulation from Embedded to High-Performance Systems[M]. Springer, US, 2010.

[8] UNDO Corp. What is Reverse Debugging, and why do we need it[EB/OL]. [2020-5-12]. http://undo.io/resources/reversedebugging-whitepaper/.

[9] FSF. GDB and Reverse Debugging . [EB/OL]. [2020-5-12]. http://www.gnu.org/software/gdb/news/reversible.html.

[10] MARK. Introducing Time Travel Debugging for Visual Studio Enterprise 2019[EB/OL]. [2020-5-12]. http://devblogs.microsoft.com/visualstudio/.

[11] Microsoft Corp. IntelliTrace Features[EB/OL]. [2020-5-12]. http://docs.microsoft.com/en-us/visualstudio/debugger/intellitrace-features.

[12] CALLAHAN R, JONES C, FROYD N, et al. Engineering record and replay for deployability: Extended technical report[J]. arXiv: 1705.05937, 2017.

[13] GITHUB. Pomp[EB/OL]. [2020-5-12]. http://github.com/junxzm1990/pomp.

[14] Microsoft Corp. Debugging Using WinDbg Preview [EB/OL]. [2020-5-12]. http://docs.micro-soft.com/en-us/windowshardware/drivers/debugger/.

[15] Microsoft Corp. Time Travel Debugging Over-view. [EB/OL]. [2020-5-12]. http://docs.microsoft.com/en-us/windowshardware/drivers/debugger/.

[16] CUI W, GE X, KASIKCI B, et al. {REPT}: Reverse Debugging of Failures in Deployed Software[C]// 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18). 2018: 17-32.

[17] MU D, GUO W, CUEVAS A, et al. RENN: Efficient Reverse Execution with Neural-network-assisted Alias Analysis[C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 924-935.

[18] HOEY J, ULIDOWSKI I, YUEN S. Reversing imperative parallel programs[J]. arXiv: 1709.00828, 2017.

[19] HOEY J, ULIDOWSKI I, YUEN S. Reversing parallel programs with blocks and procedures[J]. arXiv: 1808.08651, 2018.

[20] HOEY J, ULIDOWSKI I. Reversible imperative parallel programs and debugging[C]// International Conference on Reversible Computation. Springer, Cham, 2019: 108-127.

[21] HOEY J, LANESE I, NISHIDA N, et al. A Case Study for Reversible Computing: Reversible Debugging of Concurrent Programs[C]// International Conference on Reversible Computation. Springer, Cham, 2020: 108-127.

[22] UNDO Corp. The interactive reverse debugger for Linux-based applications[EB/OL]. [2020-5-12]. http://undo.io/solutions/products/un-dodb-reverse-debugger/.

[23] UNDO Corp. UndoDB Documentation[EB/OL]. [2020-5-12]. http://docs.undo.io/TechnicalDetails.html.

[24] Reverse Debugging with GDB [EB/OL]. (2014-4-9)[2020-5-12]. http://sourceware.org/gdb/wiki/ReverseDebug.

[25] ZHANG D F, HU X L, WANG L J. Intelligent Tracking and Debugging Technology Based on Domestic Platform[J]. Computer Systems and Applications, 2019, 28(8): 101-108.

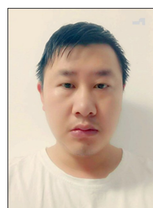
[26] KLIMUSHENKOVA A, DOVGALYUK M. Improving the performance of reverse debugging[J]. Programming and Computer Software, 2017, 43(1): 60-66.

[27] Process-Record-and-Replay[EB/OL]. [2020-5-15]. http://sourceware.org/gdb/onlinedocs/gdb/Process-Record-and-Replay.html.

[28] MOZILLA Corp. what rr does[EB/OL]. [2020-5-12]. http://rr-project.org/.

[29] GITHUB. rr[EB/OL]. [2020-5-12]. http://github.com/mozilla/rr.

[30] O'CALLAHAN R,JONES C,FROYD N,et al. Engineering record and replay for deploy-ability[C]// 2017 Annual Technical Conference ({USENIX}{ATC}). 2017:377-389.
[31] AKGUL T,MOONEY J. Assembly instruction level reverse execution for debugging[J]. ACM Transactions on Software Engineering and Methodology (TOSEM),2004,13(2):149-198.
[32] AKGUL T,MOONEY J. Instruction-level reverse execution for debugging[J]. ACM SIGSOFT Software Engineering Notes, 2002,28(1):18-25.
[33] AKGUL T,MOONEY J,PANDE S. A fast assembly level reverse execution method via dynamic slicing[C]// 26th International Conference on Software Engineering. IEEE, 2004: 522-531.
[34] LEE J. Dynamic reverse code generation for backward execution [J]. Electronic Notes in Theoretical Computer Science, 2007, 174(4):37-54.
[35] YI J. A Case for Dynamic Reverse-code Generation to Debug Non-deterministic Programs[J]. arXiv:1309. 5152,2013.
[36] GITHUB. Intel PT [EB/OL]. [2020-5-12]. https://github.com/intelpt.
[37] XU J,MU D,XING X,et al. Postmortem program analysis with hardware-enhanced post-crash artifacts[C]// 26th {USENIX} Security Symposium ({USENIX} Security 17). 2017:17-32.
[38] MU D,DU Y,XU J,et al. POMP++:Facilitating Postmortem Program Diagnosis with Value-set Analysis [J]. IEEE Transactions on Software Engineering,2019(99):1.
[39] CUI W,PEINADO M,CHA S K,et al. Retracer:Triaging crashes by reverse execution from partial memory dumps[C]// IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE,2016:820-831.
[40] GE X,NIU B,CUI W. Reverse Debugging of Kernel Failures in Deployed Systems [C] // 2020 Annual Technical Conference ({USENIX}{ATC} 20). 2020:281-292.
[41] NORDMARK E F. The fundamentals of debuggers and the challenges of Reverse Debugging [D]. Trondheim: Norwegian University of Science and Technology,2019.
[42] GURDEEP S R,TORRES L C,MARR S,et al. Multiverse Debugging: Non-Deterministic Debugging for Non-Deterministic Programs (Artifact)[J]. Dagstuhl Artifacts Series,2019,5(2): 1-3.
[43] DOVGALYUK P. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging[C]// CSMR. 2012:553-556.



XU Jian-bo, born in 1987, master. His main research interests include computer application and information security.

SHU Hui, born in 1974, Ph. D, professor. His main research interests include reverse engineering and network information security.