第 44 卷 第 4 期 计 算 机 工 程 2018 年 4 月
Vol. 44 No. 4 Computer Engineering April 2018

·安全技术· 文章编号：1000-3428(2018)04-0168-06 文献标志码：A 中图分类号：TP311

基于动态污点分析的栈溢出 Crash 判定技术

张 婧，周安民，刘 亮，贾 鹏，刘露平，贾 丽

（四川大学 电子信息学院，成都 610065）

摘 要： Fuzzing 技术和动态符号执行技术以发现程序异常为测试终止条件，却无法进一步评估程序异常的威胁严重等级。为此，阐述用于 Crash 可利用性分析的 3 种主要方法。在二进制插桩平台 Pin 上，提出一种基于动态污点分析技术的 Crash 可利用性自动化分析框架。实验结果表明，该框架能够准确有效地判断 Crash 的可利用性程度。

关键词： 软件异常；Crash 分析；可利用性判定；动态二进制插桩；动态污点分析

中文引用格式： 张 婧，周安民，刘 亮，等. 基于动态污点分析的栈溢出 Crash 判定技术[J]. 计算机工程，2018，44(4)：168-173，180.

英文引用格式： ZHANG Jing，ZHOU Anmin，LIU Liang，et al. Stack Overflow Crash Judgment Technology Based on Dynamic Taint Analysis[J]. Computer Engineering，2018，44(4)：168-173，180.

Stack Overflow Crash Judgment Technology Based on Dynamic Taint Analysis

ZHANG Jing，ZHOU Anmin，LIU Liang，JIA Peng，LIU Luping，JIA Li

（College of Electronics and Information Engineering，Sichuan University，Chengdu 610065，China）

【Abstract】 Fuzzing technology and dynamic symbol execution technology are used to discover program exceptions as the test termination condition without further assessing the threat severity of the program exception. This paper describes three main methods for Crash exploitability analysis. Based on the binary instrumentation platform Pin，this paper proposes a Crash exploitability automated analysis framework based on dynamic taint analysis technique. Experimental results show that the framework can accurately and effectively determine the exploitability extent of Crash.

【Key words】 software exception；Crash analysis；exploitability judgment；dynamic binary instrumentation；dynamic taint analysis

DOI： 10.3969/j.issn.1000-3428.2018.04.027

0 概述

随着信息化时代的到来，互联网的运用越来越广泛，尤其是计算机软件已经成为世界经济、科技、军事和社会发展的重要引擎，与此同时软件的安全问题也日益突出[1]。软件漏洞是安全问题的主要根源之一。近年来软件漏洞数量呈现明显上升趋势，软件漏洞能够引发恶性的 Web 攻击事件侵害公民权益，传播广泛的计算机病毒造成重大经济损失，实施高级可持续性攻击引发国家安全事件等[2]。因此，对软件漏洞的分析已经成为计算机领域的重要研究热点及难点。

当程序发生崩溃（Crash）时，安全研究人员需要进一步判定崩溃是由程序的内部逻辑错误引起还是由外部输入数据引起。如果是由外部输入引起，那么这很可能是一个严重的崩溃，甚至是一个可利用的漏洞[3]。因此，分析 Crash 的可利用性，即判定 Crash 是否能够被攻击者所利用以及被利用后的危害程度是否足够严重。分析崩溃的可利用性等级不仅为软件异常分析提供重要依据，还能够有效提高漏洞挖掘速度。无论对于攻击者还是防御者，这都是一项十分重要并且紧迫的工作。随着计算机软件的广泛应用，如何快速有效地分析评估 Crash 的可利用性已经成为当前漏洞挖掘与分析的关键问题之一。

本文介绍目前用于 Crash 分析领域的 3 种常见方法，利用动态二进制插桩工具 Pin 在 Windows 平台实现一种基于动态污点分析技术的 Crash 可利用性自动化判定框架，并对若干已知漏洞的应用程序进行实验验证，与 !exploitable 工具的分析结果进行对比。

1 Crash 可利用性分析方法

目前用于 Crash 可利用性分析的常见方法主要有 3 种，静态分析以 Microsoft 发布的 !exploitable 工

作者简介： 张 婧（1991—），女，硕士研究生，主研方向为信息系统安全；周安民，研究员；刘 亮，讲师；贾 鹏、刘露平，博士研究生；贾 丽，本科生。

收稿日期： 2017-06-05 **修回日期：** 2017-07-06 **E-mail：** xinyi_lan1314@163.com

具[4]为代表,静态分析与动态分析结合以二进制分析框架 BitBlaze[5]为代表,动态分析以漏洞利用自动生成框架 CRAX[6]为代表。

1.1　调用堆栈分析

2009 年微软安全工程中心基于微软在 Windows Vista 开发过程中使用 Fuzzing 的经验创建了 !exploitable工具,用于自动崩溃分析和安全风险评估。该工具通过比较调用堆栈对应的散列值分类崩溃,进而分析崩溃上下文的语义信息定义可利用性级别。它只依赖于 Crash dump,而不是导致崩溃的输入[7]。程序发生崩溃时, WinDbg 加载 MSEC. dll, 使用 !exploitable命令分析 Crash 是否可利用,首先整理所有崩溃并创建哈希值来确定崩溃的唯一性,其次查看崩溃类型并判定崩溃是否可以被恶意利用,然后分配崩溃可利用性等级:Exploitable(可利用),Probably Exploitable(可能可利用),Probably Not Exploitable(可能不可利用) or Unknown(未知)。

!exploitable工具易用性强,效率较高,可以极大地帮助安全研究人员节省分析崩溃的时间和精力。但是该工具具有较高的假阳性率,并且只能给出 Windows 平台下准确的崩溃判断,对于其他第三方软件和一些逻辑复杂的崩溃,例如堆溢出、UAF 等,准确率非常低[8]。

1.2　执行跟踪

BitBlaze 是 UC Berkely 开发的基于二进制信息的分析平台,该平台主要用于确定崩溃时哪些寄存器和内存地址来自攻击者控制的输入文件,并且切片 Null 指针查看起始位置来快速排除可利用性。该平台支持精确的分析,结合了静态和动态分析技术以及程序验证技术以满足普遍需求,有助于快速确定由基于文件变异的 Fuzzing 生成的特定崩溃是否可利用,以及确定崩溃产生的根本原因。

分析程序崩溃的根本原因,目前主要通过手动或使用调试器完成,使用 BitBlaze 可以简化和加快进程,并且提供可重复性。BitBlaze 运行在 GuestOS 的更底层,对目标程序有较好的透明性,但该平台局部开源且不便于扩展,对于污点的设置同样缺乏灵活性[9]。

1.3　端到端

2012 年 IEEE 会议提出了基于 AEG 方法改进的漏洞利用自动化框架 CRAX。它是一个基于 S2E 的符号执行平台。为了生成控制流劫持攻击,检测符号 EIP 及其他基于连续的寄存器和指针,提出了一种搜索最大连续符号内存用于有效载荷注入的系统方法。

CRAX 解决了没有源代码的大型软件系统的漏洞利用自动生成,并且可以粗粒度地确定崩溃优先级。然而 CRAX 未开源且耗时较 AEG 长,因为它进行的是整个系统的符号执行而 AEG 只是在应用程序级别,只能通过减少在 concolic 执行期间的约束数量进行时间优化。

2　相关技术

动态污点分析技术是指将非信任来源的数据进行标记,并追踪其在程序执行过程中的传递,从而达到获取关键位置与输入数据关联信息的分析方法[10]。动态污点分析能够准确地获取程序的执行过程,有效地提高污点分析的精度,错误率较低,应用性较强。动态污点分析的主要过程包括 3 个阶段,如图 1 所示[11]。

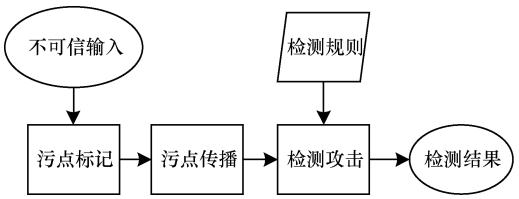

图 1　动态污点分析过程

各阶段的作用如下:

1)污点标记(污染源识别):正确识别污染的来源并进行标记。如果对象的值来源不可信,那么将该对象标记为污染的。

2)污点传播:根据正确的污点传播规则,将污点数据随着程序的运行进行污点的标记及传递(或移除)。

3)安全策略检测攻击:通过制定合理的规则与策略,确定或者检测出程序运行中存在的各种漏洞利用攻击,如缓冲区溢出等[11]。

动态污点传播分析一般采用虚拟执行或者动态二进制插桩方式对污点数据进行记录和跟踪维护[11]。动态二进制插桩是指在不影响程序正常执行结果的前提下,根据用户需要在程序动态执行过程中插入额外的分析代码以监控程序执行过程[12]。随着动态二进制插桩平台的出现,动态污点分析的实现和应用变得越加广泛。本文提出的 Crash 可利用性自动化分析框架,进行动态污点分析的前提就是需要对目标程序进行动态二进制插桩。选用动态二进制插桩平台 Pin 作为原型系统,在此基础上结合动态污点分析技术实现对 Crash 的可利用性判定。

Pin 是 Intel 公司提供的二进制程序插桩工具,支持 IA-32、Intel(R) 64 和 IA64 架构上的 Windows 和 Linux 可执行程序,具有易用、高效、可移植性以及健壮性等特点[13]。Pin 采用动态编译的方法插入探测代码,并配合使用函数内联、寄存器重分配、指令调度等优化方法,使得基于 Pin 开发的动态二进制插桩工具具有较高的运行效率[14]。Pin 工具内部含有即时编译器、虚拟机和代码缓存,插桩工具通过插桩接口函数与 Pin 进行交互[14],其整体架构如图 2所示。

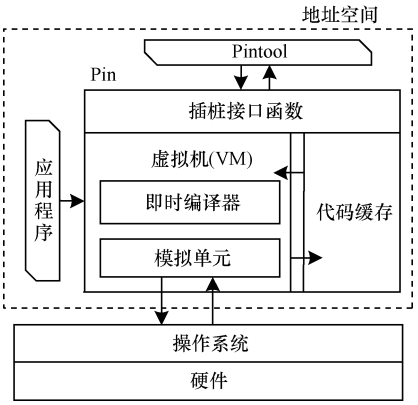

图 2 Pin 平台框架

使用 Pin 进行动态污点分析能够有效地实现自动化。Pin 的指令级插桩主要通过函数 INS_AddInstrumentFunction 来实现,它会自动在每执行一条新指令时调用回调函数 INS_InsertCall,并且在不影响原代码正常执行的同时将分析代码插入原代码序列中,进而生成一个新的代码序列并自动切换代码的控制权,使得整个分析过程自动化且高效进行[15]。使用 Pin 对目标程序进行二进制插桩的具体步骤如下:

1)调用 PIN_Init()初始化 Pin 环境。

2)调用 INS_AddInstrumentFunction(),分别声明指令级插桩函数 Instruction 和镜像级插桩函数 ModLoad。其中,函数 Instruction 中定义了一个或多个回调函数 INS_InsertCall,在每执行一条新指令时均要调用 INS_InsertCall 函数执行对当前指令的分析代码;函数 ModLoad 则对整个输入文件进行镜像级插桩。

3)调用 PIN_AddFiniFunction()声明程序退出函数 Fini,该函数用于在程序运行结束时输出最终的崩溃可利用性判断结果。

4)调用 PIN_StartProgram()启动程序。

本文以动态二进制分析平台 Pin 为基础,利用动态插桩技术和动态污点分析技术,实现了 Crash 的可利用性自动化判定框架。该框架暂不考虑保护机制的影响,如 DEP、ALSR 等。其整体架构如图 3 所示。

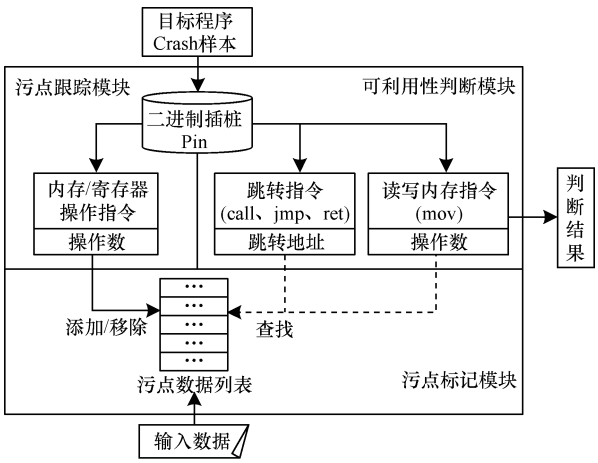

图 3 框架整体结构

3 系统设计与实现

利用该框架进行 Crash 可利用性分析的具体流程如下:

1)调用污点标记模块,将来自于不可信数据源的输入数据标记为污染源。

2)调用 Pin 对目标程序 Crash 样本进行二进制指令级插桩。

3)调用污点跟踪模块跟踪每一条内存/寄存器操作指令,判断其是否引起污点传播,并对被污染的内存和寄存器进行标记,同时修改污点数据列表。

4)调用 Crash 可利用性判定模块,对能够改变程序控制流的 call、jmp、ret 指令判断其目的地址/操作数是否已被标记,同时对进行读写内存操作的 mov 指令判断其操作的内存地址是否可控以及是否操作污点数据,由此给出判定结果。

3.1 污点标记模块

污点标记是进行污点传播分析的前提,其标记方法和结果将极大程度地影响污点传播处理和污点信息存储的效率[11]。污点标记模块主要有 2 个功能:1)将目标程序的输入数据作为污染源并为其全部添加污点标记;2)对于污点跟踪模块中的污点传播,如果有污点数据污染了内存/寄存器,则为其添加污点标记,如果已被污染的内存/寄存器被非污点数据覆盖则移除其污点标记。定义 2 个污点数据列表:addressTainted 和 regsTainted,分别用于存放被污染的内存地址和寄存器。该模块工作流程如图 4 所示。

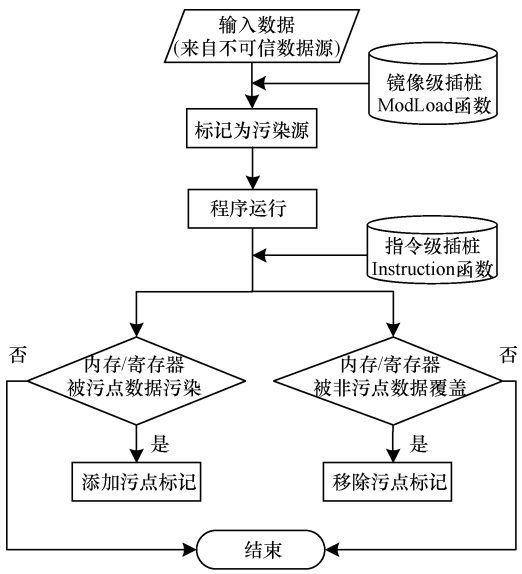

图 4 污点标记模块工作流程

3.2 污点跟踪模块

通过污点标记模块将不可信输入数据全部标记为污染源后目标程序开始执行。此时调用污点跟踪模块对程序进行指令级插桩,从而分析程序执行过程中污点的传播情况。分析时需要对每条指令做出如下判断:1)这条指令是否是内存/寄存器操作指

令;2)如果是,这条指令是否引起了污点传播。该模块工作流程如图 5 所示。

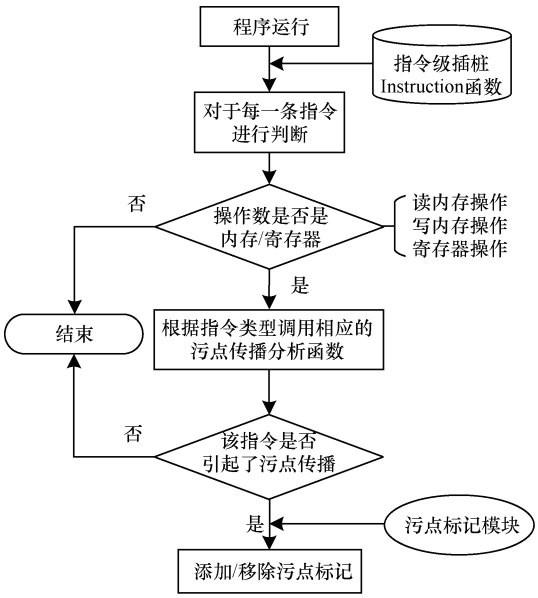

图 5　污点跟踪模块工作流程

3.3　Crash 可利用性判定模块

Crash 可利用性判定模块工作流程如图 6 所示。

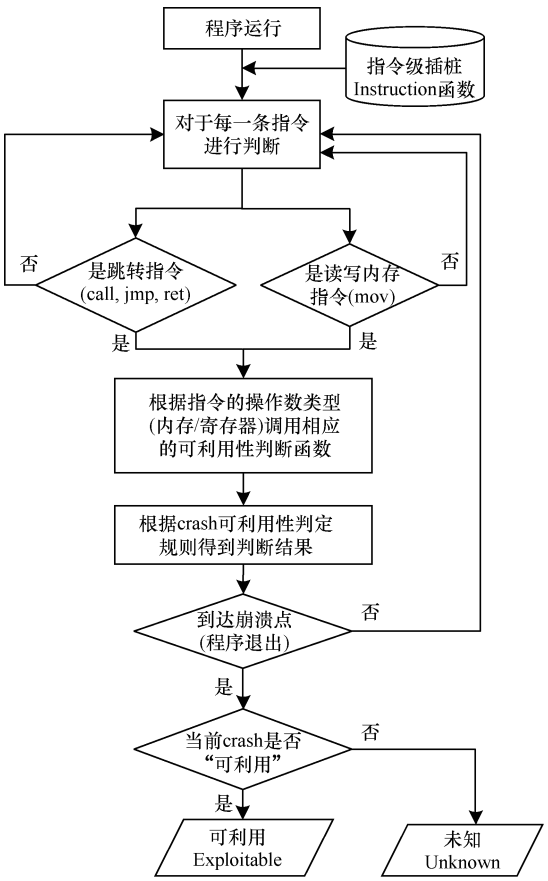

图 6　Crash 可利用性判定模块工作流程

一般情况下程序的代码段是不可重写的,攻击者只能通过修改程序的输入数据(如跳转指令的目的地址、格式化字符串函数的参数等),来控制程序的执行流从而进行攻击操作[16]。此过程中涉及到 2 类危险指令的操作,分别是跳转指令和读写内存指令,该模块通过检测在程序崩溃点之前是否出现过这 2 类危险指令而给出 Crash 可利用性的判断结果:Unknown(未知)和 Exploitable(可利用),其中,Exploitable 分为 Normal(默认)和 High。

根据上述 2 类危险指令,Crash 可利用性判定规则如下:

1)跳转指令:主要是指 call、jmp 及 ret,它们能够改变程序的控制流。程序正常运行时一般不会出现污点数据被用作跳转指令的目的地址的情况,因此当 call、jmp 或 ret 指令的目的地址是一个被污染的内存地址或寄存器时,表明攻击者很可能已经通过输入数据改写了这些目的地址,并尝试获取程序控制流来运行恶意的 shellcode 或绕过安全检查等。

(1)call/jmp 指令:判断 call、jmp 指令的目的地址是否是污点数据。对于 call/jmp 指令,其操作数可能是一个内存地址也可能是一个寄存器,分别进行如下判断:①对于 call/jmp[MEM]指令,如果[MEM]已被污染则该崩溃“可利用”,特别地,对于[MEM]为[reg_base + offset],如果 reg 或 reg 所指向的内存单元已被污染,则该崩溃“可利用”;②对于 call/jmp REG 指令,如果 REG 或者 REG 所指向的内存单元[addr]已被污染,则该崩溃“可利用”。

(2)ret 指令:判断 ret 指令的目的地址是否是污点数据并确定污点目的地址中的可控字节数。ret 指令没有操作数但会跳转到某个返回地址。首先从 esp 中获取 ret 指令的返回地址,然后进行如下判断:①如果返回地址所指向的内存单元已被污染,则该崩溃“可利用”;②如果返回地址本身已被污染(如缓冲区溢出漏洞会覆盖返回地址),则该崩溃“可利用”,此时还需要判断该污点目的地址的可控字节数,如果崩溃点之前有跳转指令,其目的地址是污点数据且 4 个字节全部可控,那么通过这条跳转指令就能得到对一个 4 GB(232 Byte)内存空间的控制权,这表明该崩溃具有极高的可利用性,因为攻击者能够在其中插入任意的 shellcode 进行攻击。

2)读写内存指令:指通过寄存器对内存单元进行读写数据操作。攻击者通常会将危险输入数据写入某块内存单元从而间接地获取控制流,该过程不可避免地会进行读写内存指令。如果读取的内存地址可控,或者对被污染的内存单元或寄存器进行了写操作甚至写入的内容是污点数据,那么这条读/写内存指令就极有可能被攻击者所利用,例如用于覆盖 SEH(结构化异常处理)[17]等。

(1)读内存指令:判断 mov REG,[MEM]指令中 MEM 是否可控。读内存指令中[MEM]的表示形式可能是[reg_base + reg_index * scale + offset],如果 reg_base 或 reg_index 已被污染,进而获取 ebp 的内存

地址,如果 ebp 的内存地址也被污染,则该崩溃“可利用”。因为如果[MEM]的 reg_base 和 reg_index 中任意一个被污染,则内存地址可控,也就意味着攻击者能够通过读内存指令将某个指定地址中的值写入 REG 中,从而触发读内存异常或引起程序崩溃甚至用于覆盖 SEH,此时进一步判断 ebp 的内存地址是否被污染,如果是则表明栈中的 SEH 可能已被修改,则该崩溃“可利用”。

(2)写内存指令:判断 mov[MEM],REG 指令中 REG 是否是污点数据,MEM 是否可控。写内存指令中[MEM]的表示形式可能是[reg_base + reg_index * scale + offset],如果 reg_base 或 reg_index 已被污染,进而获取 ebp 的内存地址,如果 ebp 的内存地址也被污染则该崩溃“可利用”。因为如果[MEM]的 reg_base 和 reg_index 中任意一个被污染,则内存地址可控,若此时 REG 也是污点数据,这就意味着攻击者能够通过写内存指令将任意数值通过 REG 写入某个指定地址的值中,从而触发读内存异常或引起程序崩溃甚至用于覆盖 SEH,此时进一步判断 ebp 的内存地址是否被污染,如果是则表明栈中的 SEH 可能已被修改,则该崩溃“可利用”。

4 实验结果分析

本文的实验对象为存在已知漏洞的 CoolPlayer、Word 等多个应用程序,实验环境如表 1 所列,实验结果如表 2 所列。其中,* 表示标注为! exploitable 工具的测试结果。以 1 号实验为例,当 CoolPlayer Portable 2.19.1 试图打开一个包含超过 260 Byte 的特定字符串的.m3u 文件时,则会触发栈溢出漏洞。构建引发该软件崩溃的输入数据如图 7 所示。

表 1 测试环境参数

名称	相关信息
处理器	Intel(R) Xeon(R) CPU E3-1231 v3@3.40 GHz
内存	4 GB
操作系统	Windows 7 Ultimate,64 bit 6.1.7600
虚拟机	VMware Workstation 11.1.0 build-2496824
DBI	pin-2.10-45467-msvc10-ia32_intel64-windows
编译器	Microsoft Visual Studio 2010 professional, x86 v10.0.30319.1
调试工具	WinDbg 6.11.0001.404,x86 WinDbg 10.0.14321.1024,AMD64
! exploitable	MSECExtensions 1.6.0,x64

表 2 实验结果对比

序号	被测程序	漏洞编号	漏洞类型	污点源/Byte	耗时/s	Crash 指令	测试结果
1	CoolPlayer Portable 2.19.1	CVE-2009-1437	栈溢出 覆盖返回地址	362	61	ret * ret	Exploitable_High * Exploitable
2	Soritong 1.0	CVE-2009-1643	栈溢出 覆盖 SEH 和返回地址	276	48	ret * ret	Exploitable * Exploitable
3	PHP 7.0.0	CVE-2015-8617	格式化字符串	109	32	ret * mov	Exploitable * Exploitable
4	Word 2003 11.0.8324.0	CVE-2010-3333	栈溢出 覆盖返回地址	46	102	mov * mov	Exploitable * Probably Not Exploitable
5	暴风影音 2012 3.10.4.8	CNVD-2010-00752	栈溢出 覆盖 SEH	64	20	ret * mov	Unknown * Probably Exploitable
6	Adobe Flash Player 10.2.159.1	CVE-2011-2130	缓冲区溢出	128	16	mov * mov	Exploitable * Unknown

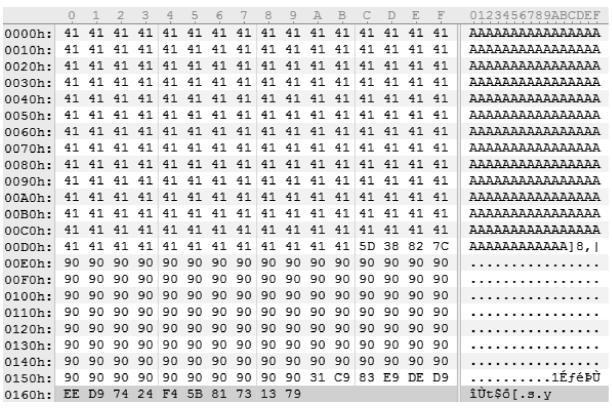

图 7 污染源文件

使用该框架和! exploitable 工具分别进行崩溃可利用性判定的结果如图 8 所示。

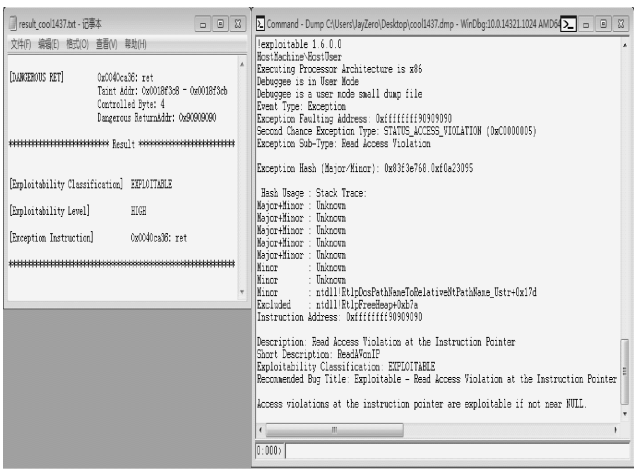

图 8 测试结果对比

该框架不仅能够准确定位到发生崩溃的指令,还能检测到引发崩溃的 ret 指令的返回地址被覆盖为用户输入数据,且根据返回地址中的可控字节数对该崩溃可利用性的高低进行进一步判定,准确度较!exploitable 工具更高。

对表 2 的实验结果分析如下:

1)该框架能够准确定位到发生 Crash 的指令,并进一步判定可利用性高低,针对栈溢出类型漏洞的崩溃可利用性判定,相比!exploitable 工具准确率较高。

2)当测试程序小巧且污染源文件较小时,该框架分析速度较快。但对于 Word 等较大程序,尽管污染源文件很小,该框架分析速度也明显下降。

3)针对堆溢出、UAF、格式化字符串以及逻辑关系复杂的崩溃,该框架暂时无法准确有效地判定其可利用性。

4)动态污点分析针对数据流,并不能对控制流信息进行分析,往往难以发现与控制流相关的脆弱性,使得最终分析结果存在精度不高的问题。

5　方法比较

比较该框架与前文所提到的 3 种分析方法,如表 3 所示。

表 3　对比分析

对比项	本文框架	! exploitable	BitBlaze	CRAX
主要方法	污点分析/动态插桩	散列算法	污点分析/符号执行	符号执行
对象	二进制程序	崩溃转储	二进制程序	应用程序
动/静态	动态	静态	动静结合	动态
针对漏洞	栈溢出	栈溢出	缓冲区溢出	栈/堆溢出
是否开源	未开源	开源	局部开源	未开源
平台	Windows	Windows	Linux	Linux/Windows
主要优点	易用/栈溢出分类准确	适用崩溃数量巨大/易用性强	加快崩溃原因分析	无源码的软件系统漏洞利用生成
主要缺点	效率较低/堆溢出等分类不准确	假阳性率较高/堆溢出等分类不准确	效率低下/无法分析无效读取崩溃	耗时较长

综合上述分析可知,BitBlaze 和 CRAX 这 2 种分析工具由于尚未完全开源,应用覆盖范围较窄,但对于程序崩溃可利用性判定研究具有重要的参考价值。目前主要利用!exploitable 工具静态分析 Crash dump 的方法初步判定 Crash 可利用性,然而该工具具有较高的假阳性率。本文提出的框架利用动态污点分析技术,便于扩展,针对栈溢出漏洞的可利用性判定准确率明显高于!exploitable 工具。

6　结束语

本文采用 Windows 下二进制插桩平台 Pin,提出了一种基于动态污点分析技术的 Crash 可利用性自动化判定框架。该框架能够有效地检测出可利用的崩溃并分配可利用性等级,准确分类栈溢出漏洞。但该框架主要存在 2 个问题:1)对于逻辑关系较为复杂的崩溃只能进行粗粒度的判断,准确度较低;2)对于较大的程序或样本,检测时间开销较大且收集的信息易重复,效率降低。

下一步将针对该框架的性能和效率进行优化,增加对程序指令的识别覆盖面,完善 Crash 可利用性判断规则,避免污染缺失的情况,提高判定准确性;通过对内存读写操作代码的优化,提高运行效率;结合动态符号执行技术[18]分析异常指令后的多条路径,提高路径覆盖率。

参考文献

[1] 吴世忠,郭　涛,董国伟,等.软件漏洞分析技术进展[J].清华大学学报(自然科学版),2012,52(10):1309-1319.

[2] MCGRAW G. Software security: building security in[M]. Boston, USA: Addison-Wesley Professional, 2006.

[3] KROHNHANSEN H. Program crash analysis: evaluation and application of current methods[EB/OL]. [2017-06-05]. https://www-esv. nhtsa. dot. gov/Proceedings/24/files/24ESV-000055. PDF.

[4] Microsoft. ! exploitable crash analyzer——MSEC debugger extensions[EB/OL]. [2017-06-05]. http://msecdbg. codeplex. com.

[5] MILLER C, CABALLERO J, BERKELEY U, et al. Crash analysis with BitBlaze[J]. Revista Mexicanade Sociología, 2010, 44(1): 81-117.

[6] HUANG Shih-kun, HUANG Min-hsiang, HUANG Po-yen, et al. CRAX: software crash analysis for automatic exploit generation by modeling attacks as symbolic continuations[C]//Proceedings of the 6th International Conference on Software Security and Reliability. Washington D. C., USA: IEEE Press, 2012: 78-87.

[7] WEINSTEIN D, SHIRK J. The history of the ! exploitable crash analyzer [EB/OL]. [2017-06-05]. http://blogs. technet. com/b/srd/archive/2009/04/08/the-history-of-the-exploitable-Crash-analyzer. aspx.

[8] ZHANG Puhan, WU Jianxiong, XIN Wang, et al. Program crash analysis based on taint analysis[C]//Proceedings of the 9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. Washington D. C., USA: IEEE Press, 2014: 492-498.

(下转第 180 页)

参考文献

[1] ADI S. Identity-based cryptosystems and signature schemes[J]. Lecture Notes in Computer Science, 1984, 21(2): 47-53.

[2] XUN Yi. An identity-based signature scheme from the weil pairing[J]. IEEE Communications Letters, 2003, 7(2): 76-78.

[3] BARRETO P S L M, MCCULLAGH N, QUISQUATER J J. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps[C]//Proceedings of International Conference on Theory and Application of Cryptology and Information Security. Washington D. C., USA: IEEE Press, 2005: 515-532.

[4] BELLARE M, NAMPREMPRE C, NEVEN G. Security proofs for identity-based identification and signature schemes[M]. Berlin, Germany: Springer, 2004.

[5] KUROSAWA K, HENG S H. From digital signature to ID-based identification/signature[C]//Proceedings of International Workshop on Public Key Cryptography. Berlin, Germany: Springer, 2004: 248-261.

[6] DAN B, BEN L, HOVAV S. Short signatures from the weil paring[J]. Journal of Cryptology, 2004, 17(4): 297-319.

[7] CORON J S. Optimal security proofs for PSS and other signature schemes[C]//Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques. Berlin, Germany: Springer, 2002: 272-287.

[8] GE S. Tight proofs for signature schemes without random Oracles[C]//Proceedings of International Conference on Theory and Applications of Cryptographic Techniques. Berlin, Germany: Springer, 2011: 189-206.

[9] 卢 超, 钱海峰. 标准模型下的在线/离线多签名方案[J]. 计算机应用研究, 2010, 27(9): 3514-3517.

[10] 胡国政, 洪 帆. 标准模型中可证安全的签名方案[J]. 武汉理工大学校报, 2009, 31(15): 130-134.

[11] POINTCHEVAL D, STERN J. Security arguments for digital signatures and blind signatures[J]. Journal of Cryptology, 2000, 13(3): 361-396.

[12] 刘振华, 张襄松, 田绪安, 等. 标准模型下基于身份的具有部分消息恢复功能的签名方案[J]. 北京工业大学学报, 2010, 36(5): 654-658.

[13] WANG Z, CHEN H. Emerging directions in embedded and ubiquitous somputing[M]. Berlin, Germany: Springer, 2007.

[14] BELLARE M, ROGAWAY P. The exact security of digital signatures: how to sign with RSA and Rabin[C]//Proceedings of International Conference on Theory and Applications of Cryptographic Techniques. Berlin, Germany: Springer, 1996: 399-416.

[15] MAURER U M, WOLF S. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms[J]. SIAM Journal on Computing, 1998, 28(5): 1689-1721.

[16] SCHNORR C P. Efficient identification and signatures for smart cards[M]. Berlin, Germany: Springer, 1989.

[17] LIBERT B, QUISQUATER J J. The exact security of an identity based signature and its applications[EB/OL]. [2016-11-25]. https://eprint.iacr.org/2004/102.pdf.

[18] ABE M, OKAMOTO T. A signature scheme with message recovery as secure as discrete logarithm[J]. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, 2001, 84(1): 197-204.

编辑 吴云芳

(上接第 173 页)

[9] 叶永宏, 武东英, 陈 扬. 一种基于细粒度污点分析的逆向平台[J]. 计算机工程与应用, 2012, 48(28): 90-96.

[10] 史大伟, 袁天伟. 一种粗细粒度结合的动态污点分析方法[J]. 计算机工程, 2014, 40(3): 12-17.

[11] 宋 铮, 王永剑, 金 波, 等. 二进制程序动态污点分析技术研究综述[J]. 信息网络安全, 2016(3): 77-83.

[12] RODRÍGUEZ R J, ARTAL J A, MERSEGUER J. Performance evaluation of dynamic binary instrumentation frameworks[J]. IEEE Latin America Transactions, 2015, 12(8): 1572-1580.

[13] REDDI V J, JANAPA V, SETTLE A, et al. PIN: a binary instrumentation tool for computer architecture research and education[C]//Proceedings of Workshop on Computer Architecture Education. New York, USA: ACM Press, 2004: 1-7.

[14] 王 乾. 基于动态二进制分析的关键函数定位技术研究[D]. 郑州: 信息工程大学, 2012.

[15] 孔德光, 郑 烇, 帅建梅, 等. 基于污点分析的源代码脆弱性检测技术[J]. 小型微型计算机系统, 2009, 30(1): 78-82.

[16] 黄 昭. 一种改进的动态污点分析模型[D]. 武汉: 华中科技大学, 2011.

[17] WU Weimin, GUO Chaowei, HUANG Zhiwei, et al. Vulnerability exploitation technology of structured exception handling based on windows[J]. Computer Engineering, 2012, 38(20): 5-8.

[18] ZHANG Yufeng, CHEN Zhenbang, WANG Ji, et al. Regular property guided dynamic symbolic execution[C]//Proceedings of the 37th IEEE International Conference on Software Engineering. Washington D. C., USA: IEEE Press, 2015: 643-653.

编辑 顾逸斐