面向动态生成代码的攻防技术综述

吴 炜, 霍 玮, 邹 维

中国科学院信息工程研究所 北京 中国 100093
中国科学院网络测评技术重点实验室 北京 中国 100195
网络安全防护技术北京市重点实验室 北京 中国 100195
中国科学院大学 北京 中国 100049

摘要 动态代码生成技术广泛使用在浏览器、Flash 播放器等重要日常软件中, 近年来其中曝出严重的安全问题, 为控制流劫持攻击和相应的防御提供了新机会, 受到越来越多的关注。针对动态生成代码在数据区且可被执行和直接依赖输入的特性, 本文从代码注入攻击和代码重用攻击两个角度总结分析了控制流劫持攻击新技术, 并从强制性防御和闪避防御(Moving target defense)两个角度对相关的主要防御新方法进行了阐述。同时提出动态代码生成系统安全性的衡量模型, 对代表性防御技术进行对比分析和评估, 并探讨了面向动态生成代码攻防技术的发展趋势和下一步的研究方向。

关键词 软件安全; 即时编译; 动态二进制翻译; 控制流劫持; 防御技术
中图法分类号 TP 309.1 **DOI 号** 10.19363/j.cnki.cn10-1380/tn.2016.04.005

Survey on Attacking and Defending Technologies of Dynamic Code Generation

WU Wei, HUO Wei, ZOU Wei

Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences, Beijing 100195, China
Beijing Key Laboratory of Network Security and Protection Technology, Beijing 100195, China
University of Chinese Academy of Sciences, Beijing 100049, China

Abstract Dynamic code generation (DCG) is a technique widely deployed in important daily software such as web browser and Flash player. The threat posed by dynamic code generation has gained more and more attention in recent years because it provides new opportunities for control flow hijacking attack. In this paper, we summarize new control flow hijack attacks against DCG based on the paradigm that DCG generates executable code according to input program on-the-fly in two categories: code injection attack and code reuse attack. We systematically present the defense mechanisms for DCG in two categories: enforcement-based defense and moving target defense. We propose a model to evaluate existing defense technology based on defense benefit and defense cost. We also analyze the developing trend of attacking/defending technologies on dynamic code generation and give some suggestions on future research.

Key words software security; JIT compilation; dynamic binary translation; control flow hijack; defense mechanism

1 引言

浏览器、Flash 播放器、虚拟机和模拟器已成为互联网时代广泛使用的基础性软件, 它们普遍采用即时编译(Just-In-Time compilation)和动态二进制翻译(Dynamic binary translation)等动态代码生成(Dynamic code generation)技术。该技术可在软件系统运行过程中动态产生或修改可执行代码[1-5], 称为动态生成代码(Dynamic generating code)。

控制流劫持[6]是一种经典且有效的攻击方法, 攻击者通过改变目标程序正常的执行流程并进行控制从而达到攻击的目的。控制流劫持的方法分为两大类: 注入式和非注入式。传统的注入式攻击[7]需要同时满足三个攻击条件: 1)掌握可注入攻击代码的漏

通讯作者: 霍玮, 博士, 副研究员, Email: huowei@iie.ac.cn。
本课题得到中国科学院百人计划[人字(2013)46 号]、北京市科委重点项目课题“行业场景构建与漏洞分析关键技术研究”(D161100001216001)和中国科学院战略性先导科技专项“重点行业应用系统信息安防关键技术研究”(XDA06010703)资助。
收稿日期: 2016-06-15; 修改日期: 2016-08-03; 定稿日期: 2016-10-10

洞; 2)控制程序原有执行流程并成功跳转到攻击代码起始处; 3)攻击代码可被执行。非注入式攻击, 如代码重用攻击(Code reuse attack) [8,9], 降低了传统注入式攻击的攻击条件。通过重用目标程序中的工具代码片段(Gadget)构造攻击代码, 既避免注入攻击代码又可保证攻击代码的可执行性。但可被利用的代码数量和串接难度成为代码重用攻击有效的关键因素。然而, 利用动态生成代码, 既不需依赖漏洞注入攻击代码, 也不需依赖程序中是否存在合法的攻击代码段, 仅通过精心设计程序输入就可以完成攻击代码注入或者攻击代码段构造, 同时可以绕过数据执行保护(Data execution prevention)和地址随机化(Address space layout randomization)等防御技术, 大大降低了控制流劫持的技术难度。

动态代码生成技术使得传统代码安全防护技术的有效性也面临着挑战。由于攻击代码可与动态生成的正常代码存储在同一数据区, 数据执行保护无法阻止攻击代码的执行。可执行代码在运行时被动态生成, 因此依赖静态分析与静态插装的控制流完整性(Control flow integrity)[10]保护难以实施, 基于静态分析和静态二进制变换的软件多样化(Software diversification)[11]技术也同样难以应用。

围绕动态生成代码的攻防对抗新技术和新方法成为近年来的研究热点。2010 年 Blazakis 在 Black Hat 会议上首先提出了针对 Flash 播放器中动态生成代码的攻击方法[12], 随后这种攻击方法被扩展到各种动态代码生成系统。安全研究人员相继在 Pwn2Own、Black Hat 和 NDSS 等顶级会议上提出了面向动态生成代码的一系列新攻击方式[13-18]。同时, 研究人员也提出了许多针对动态生成代码的新防御技术[18-28]。本文围绕动态生成代码, 从攻击与防御两方面对近年来的研究成果进行总结归纳, 概述技术的基本原理和应用效果, 并建立适用于面向动态生成代码攻防技术的衡量模型, 对代表性技术进行对比分析和评价, 通过分析潜在的攻击手段, 指出提高动态生成代码安全性的发展方向。本文的主要贡献: 1)分类阐述针对动态代码生成技术的攻击技术; 2)分类论述加固动态代码生成系统的防御技术的原理和研究进展; 3)提出面向动态生成代码系统的安全度量模型, 并基于该模型对防御技术进行了对比分析; 4)对基于动态生成代码的攻防技术的发展趋势进行了展望。

本文首先对动态生成代码及相关攻防技术的背景和术语进行了介绍(第 2 节), 然后对动态生成代码的原理进行了概括, 揭示动态生成代码的特性(第 3 节)。从代码注入攻击和代码重用攻击两个角度介绍了动态代码生成技术为攻击者带来的新机会(第 4 节)。从强制性防御和闪避防御两个角度阐述了针对动态代码生成的防御技术的原理、挑战和研究进展(第 5 节)。基于防御成本和防御效果, 衡量了动态代码生成系统的安全性(第 6 节), 最后总结并给出了研究发展建议。

2 研究背景

动态生成代码主要存在于即时编译系统和动态二进制翻译系统中。即时编译系统广泛存在于电脑和移动终端, 可显著提高动态语言解释执行的效率。随着 JavaScript 语言广泛使用于动态网页中[68], 当前主流浏览器, 如 Chrome、Safari、IE、Firefox 等都对 JavaScript 程序提供支持, 并采用了即时编译技术提高 JavaScript 的执行效率[1,2], 如在特定情况下可使 Google V8 引擎速度提高 12 倍。Adobe Flash 播放器几乎安装在所有的电脑和移动终端, 其也采用即时编译技术加速 ActionScript 的解释执行来为用户带来更好的多媒体体验。此外, JVM 和.NET CLR 等虚拟机均使用了即时编译技术。动态二进制翻译[29]广泛使用在跨平台支持[5,30]、遗产软件移植、动态程序分析[31]和安全防御[32,33]中以增强可移植性与安全性。二进制翻译技术作为虚拟化软件 Qemu[5]的核心技术, 随着云计算的发展而被广泛使用。由于动态生成代码直接受控于软件输入并存储于内存堆区, 传统的攻防技术, 如控制流劫持、数据执行保护、内存完整性检测、闪避防御等在应用原理、应用场景和应用效果等方面都面临新的挑战, 面向动态生成代码的攻防技术和方法成为新的研究热点。

在针对动态生成代码的攻击技术中, 控制流劫持攻击是研究的热点。控制流劫持攻击通过劫持函数返回地址或代码指针将控制流跳转到攻击者指定的代码地址并执行, 从而完成攻击。根据攻击代码来源的不同, 又可进一步分为代码注入攻击和代码重用攻击。在代码注入攻击中, 需要利用目标程序漏洞将包含 Shellcode 的代码注入到内存可执行区域并让 Shellcode 得到执行机会。代码重用攻击则利用程序中现有“合法”代码, 绕过数据执行保护完成攻击, 返回导向编程(Return oriented programming, 简称 ROP)[8]攻击是其中的典型。ROP 寻找程序内部的工具代码片段组合形成 ROP 链(ROP chain)。工具代码片段是以返回指令结尾的若干连续指令, 将多段工具代码片段精心组合可以构造等效于 Shellcode 的 ROP 链直接完成攻击, 或者仅通过构造 ROP 链调用 VirtualProtect 等函数关闭 Shellcode 所在页面的数据

执行保护, 进而执行 Shellcode 间接完成攻击。文献[34]对 ROP 的利用条件进行了建模分析。为使用 ROP 技术, 需要利用目标程序漏洞控制程序执行流程, 依次执行工具代码片段完成攻击。

数据执行保护和地址空间随机化是广泛应用于现代操作系统中的两种防御技术。数据执行保护将内存页设置为可执行的代码段或不可执行的数据段, 一种常见的数据执行保护“W ⊕ X”可以防止内存页被同时设置为可写和可执行, 能够有效防御传统代码注入攻击。地址空间随机化将程序、链接库的加载位置和堆、栈的起始地址随机化, 使攻击者难以预测目标指令的地址位置, 很大程度上提高了攻击的门槛。然而, 地址随机化也有兼容性、熵值小和易受到信息泄露攻击的缺点。一些程序和动态链接库不兼容地址随机化保护, 容易被 ROP 攻击直接利用。在 32 位操作系统中, 地址随机化为堆提供的随机性不足, 低熵的随机化易受到暴力枚举的攻击[35]。堆喷射(Heap spray)是绕过 32 位地址随机化的有效方式。同时, 研究者发现了利用程序漏洞泄露内存地址从而绕过地址随机化的方式[36-39]。

控制流完整性[10,41-43]保护是一类通过检查控制流非正常跳转从而防止程序控制流被劫持的防御方法。但传统控制流完整性防护技术只能保护静态编译产生的代码。闪避防御技术(Moving Target Defense)[44]是一种新兴的防御技术, 其中包括软件多样化技术[11]。将软件多样化技术应用到动态代码生成上可以增加生成代码的不确定性, 生成多变的不同形态的但功能相同的代码, 减少攻击者对于生成代码的知识, 从而降低攻击成功的概率[24,25]。

3 动态代码生成技术原理

动态代码生成技术能够在运行时新产生并执行优化的本地机器码, 被广泛应用于即时编译系统和动态二进制翻译系统中以提高系统性能、可移植性和安全性。尽管该技术在不同的动态代码生成器中实现方式不同, 但其基本行为特征是相同的, 都是对输入程序代码, 在运行时直接翻译生成本地机器码执行或动态优化已生成的本地机器码并执行。

现代代码解释器中广泛使用了动态代码生成技术。解释型语言通常会将程序编译为其自定义的字节码并执行。当字节码在解释器虚拟机中运行时, 即时编译器开始为其动态生成语义等价且运行速度更快的可执行代码, 通常生成宿主机的本地机器码。

一个具体的例子如图 1 所示, 在上层语言 ActionScript 中的一段源码被动态翻译成了对应的本地代码。

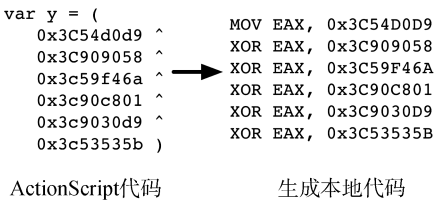

图 1 动态代码生成实例

软件动态翻译器对动态生成代码主要有四类操作: 生成、修改、删除和调用。如图 2 所示, 软件动态翻译器[18]读入待翻译的源程序, 优化编译器以函数为单位根据输入程序生成本地代码到堆上的代码缓存(Code cache, 又称 Code heap[42]、JIT buffer[15])中, 并通过执行器调用动态生成代码。对于输入程序的某一个函数, 软件动态翻译器生成的本地代码的语义与输入函数应当是等价的。由于本地代码会在运行时生成并被调用执行, 因此代码缓存区域需被设置为可写且可执行。优化编译器还可以收集轮廓信息修改动态生成代码, 进行代码优化。垃圾回收器对代码缓存进行自动内存管理, 对动态生成代码进行删除。

因此, 动态生成代码具有两个特性: 1)直接受控于输入; 2)存储于可执行的数据区。

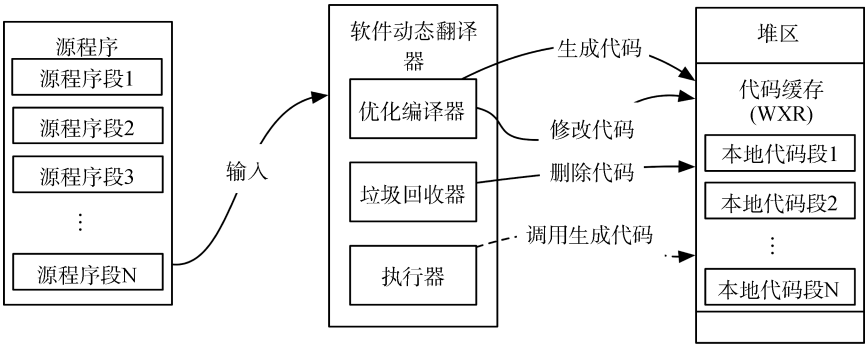

图 2 动态代码生成技术基本框架

4 动态代码生成的攻击威胁

控制流劫持漏洞是一类危害严重的漏洞。控制流劫持攻击是一种最常见的利用控制流劫持漏洞进行软件攻击的手段, 可导致攻击者获得任意代码执行能力从而安装木马、病毒, 对计算机和网络系统的机密性、可用性和完整性造成破坏。控制流劫持攻击一般可分为代码注入攻击(Code injection attack)和代码重用攻击[6]。传统代码注入攻击可以被数据执行保护防御, 但是动态生成代码位于可执行的数据区, 因此向代码缓存注入的恶意代码可以绕过数据执行保护。同时, 动态生成代码直接受控于输入的特性使攻击者可以操纵输入程序生成所需的工具代码, 更易于实施代码重用攻击。动态代码生成提供了新的攻击方式, 近年来研究者们发现许多利用动态生成代码进行攻击的机会[12,16-18]。

4.1 控制流劫持漏洞

控制流劫持攻击的必要条件是存在控制流劫持漏洞。出于性能考虑, 动态代码翻译器(如即时编译器)往往用 C 或 C++语言编写。由于动态代码翻译器的复杂性, 代码量往往高达数十万行甚至百万行。大规模、复杂的 C 或 C++程序中内存安全存在更大的隐患。各类溢出漏洞、释放后重用漏洞、类型混淆漏洞等均可能被用来劫持程序原有的控制流。如 Google V8 JavaScript Engine 中存在堆溢出漏洞(CVE-2009-2555), 可被用来劫持程序的控制流, 跳转到攻击代码执行。根据攻击代码来源的不同, 又可进一步分为代码注入攻击和代码重用攻击。

4.2 代码注入攻击

传统代码注入攻击利用目标程序的输入信道将实现恶意功能的 Shellcode 注入程序内存空间中并通过控制流劫持漏洞执行。数据执行保护通过设置数据页面不可执行能够阻止 Shellcode 执行, 很大程度上提高了代码注入攻击的门槛。但在动态代码生成技术中, 代码缓存本身就需要写和执行权限, 使得数据执行保护失效, 从而为攻击者注入执行 Shellcode 提供了便利。从注入方式的角度, 攻击者可以在利用代码缓存覆盖攻击(利用内存任意写漏洞向代码缓存注入恶意代码)这种传统方式之外, 不利用内存漏洞, 仅需构造畸型源程序, 就迫使软件动态翻译器生成恶意代码(如 JIT 喷射攻击)。

堆喷射是一种利用堆操作漏洞向堆上注入包含 Shellcode 的大量重复指令, 从而绕过地址随机化的注入攻击方法。数据执行保护是防御堆喷射的传统方法, 因为在这种情况下攻击者需要先利用 ROP 等代码重用技术将数据执行保护关闭(如调用 VirtualProtect 或 mprotect 函数), 才能执行注入的 Shellcode, 增加了漏洞利用的难度。此外, Egele 等还提出一种在字符串缓冲区中发现 Shellcode 的堆喷射防御技术[46]。针对上述防御方法, 攻击者提出了一种向代码缓存注入 Shellcode 的简便方法, 即 JIT 喷射攻击(JIT Spraying)。JIT 喷射攻击是由堆喷射演进而来的, 利用 JIT 动态生成本地机器码的特性完成代码注入攻击。由于 Shellcode 被直接注入到代码缓存中, 使得数据执行保护和 Egele 等的防御方法均失效。JIT 喷射攻击最早由 Blazakis 在 2010 年 Black Hat 上提出, 可以绕过地址随机化和数据执行保护攻击 Adobe Flash 的即时编译器[12]。JIT 生成代码存放在可读写并可执行的代码缓存中, JIT 喷射攻击强制即时编译器产生大量代码进行堆喷射攻击以绕过地址随机化防御。JIT 喷射攻击的原理如图 3 所示。攻击者利用包含立即数的脚本指令, 通过控制源程序将“常量”作为立即数注入即时编译器的生成代码中(如指令“*XOR EAX 0x3c909090*”), 然后通过一个控制流劫持漏洞跳转到“常量”的地址加一个偏移值, 此时“常量”就变成了可以被识别执行的代码片段。通过连续输入大量的常量就可以在代码缓存中生成攻击者需要的 Shellcode。已公开的 JIT 喷射攻击包括针对 Safari 浏览器 JavaScript 即时编译器的攻击[45]和对 Adobe Flash 的攻击。

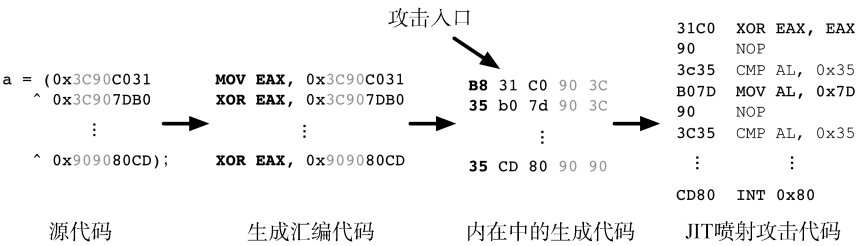

图 3 JIT 喷射攻击[25]

4.3 代码重用攻击

传统 ROP 攻击首先在程序及其依赖库中寻找工具代码片段, 再在运行时采用绕过地址随机化的技术获得工具代码片段的内存地址, 如利用一个地址

泄露漏洞获取某个动态链接库的基址, 最后劫持控制流跳转到工具代码片段执行。动态代码生成技术能够在堆上动态输出本地机器码, 且输出内容可以由高层编程语言所构成的输入所控制。这意味着攻击者可以生成自己需要的工具代码片段用于 ROP 而不依赖于程序和动态加载库中已存在可利用的工具代码片段, 同时也使 G-free 编译[48]等 ROP 攻击缓解技术失效。Serna 在报告[16]中提出了一种向代码缓存区喷射用于泄露内存地址的工具代码片段的攻击方式。动态代码生成为 ROP 等代码重用攻击提供了便利。

一种针对动态代码生成的 ROP 攻击可以分为以下三个步骤: 1)控制动态代码翻译器的输入源程序以在代码缓存产生所需工具代码片段; 2)通过信息泄露定位代码缓存区中工具代码片段的位置; 3)将控制流跳转到工具代码片段开始执行。为完成该过程, 需要攻击者深刻理解目标软件的防御机制并利用目标软件中的信息泄露漏洞。Athanasakis 等人提出一种利用 Firefox 和 IE 浏览器中 JavaScript 即时编译器动态生成工具代码片段, 并结合信息泄露漏洞进行代码重用攻击的方法[15]。Snow 在文献[40]中提出 JIT-ROP 方法, 可在上述第二步中应用。该方法利用一个信息泄露漏洞读取大量内存内容并在其中搜索 ROP 所需的工具代码片段, 即时构建 ROP 完成攻击。

5 动态生成代码的安全防御

软件漏洞是导致控制流劫持攻击的根本原因。研究者提出了一系列控制流劫持漏洞的挖掘技术[49-52]。但是由于漏洞挖掘本身的难度, 不能确保发现所有控制流劫持漏洞, 因此研究人员提出了漏洞去效技术(vulnerability neutralization), 如 Windows 平台的 EMET[66]和 Linux 平台的 grsecurity[67]等, 阻止攻击发生。漏洞去效是指通过部署各种防御机制, 使得软件系统中的漏洞无法被攻击者成功利用的一类防御技术, 典型方法包括数据执行保护、控制流完整性保护和代码多样化等。研究者将上述方法应用到动态代码生成程序中, 提出了若干改进技术[18-25]。本文根据防御技术的特点, 分强制性防御和灵敏变换攻击面的闪避防御两类, 对新提出的漏洞去效方法进行介绍和分析。

5.1 强制性防御

强制性防御技术(enforcement-based defenses[11])旨在为程序执行过程中可能被攻击者利用的部分添加防御, 从而防止对此部分的利用。数据执行保护是一种典型的强制性防御技术, 该技术通过防止数据段被执行或者代码段被改写提高了传统代码注入攻击的难度, 但是, 攻击者依然可以利用合法的代码片段进行代码重用攻击。另一种典型的强制性防御技术是控制流完整性保护, 该技术用于阻止现实攻击中常见的劫持控制流行为, 并且能够防御代码重用攻击。控制流完整性有效的前提条件是代码完整性, 即代码段不能被篡改, 但动态生成代码位于可写数据区, 因此攻击者可能利用内存任意写漏洞破坏动态生成代码的完整性以绕过控制流完整性保护。在实际中, 控制流完整性和数据执行保护通常组合使用以有效防御代码注入攻击和代码重用攻击。本节分别从数据执行保护和控制流完整性两个方面综述针对动态生成代码的强制性防御新技术。

5.1.1 数据执行保护

数据执行保护是防御代码注入攻击的重要防御手段。目前主流的数据执行保护通过禁止内存页被同时设置为可写和可执行, 阻止攻击者执行注入数据段的 Shellcode 或篡改代码段, 通常记为 W⊕X。动态代码生成需要既可写又可执行的内存空间以存放或修改动态产生的代码, 与现有数据执行保护实现机制无法兼容。因此, 研究者提出了基于权限转换的数据执行保护[19,26]和基于权限分离的数据执行保护[18,20,28]两类技术, 实现强制性防御。

代表性的基于权限转换 W⊕X 方法由南京大学陈平等人[19]提出, 其原理如图 4 所示。该方法在即时编译器的不同工作阶段, 对所使用的内存区域设置不同的权限, 在生成本地代码时将代码缓存设置为可写不可执行, 在开始执行生成代码前将代码缓存页面设置为不可写和可执行, 从而防御 JIT 喷射攻击。因为即时编译器比代码执行器更有可能出现控制流劫持漏洞[20], 所以该方法有效缩短了攻击者的攻击窗口, 能够对当时的野外 JIT 喷射攻击实例进行防御且具有可忽略的运行时开销。

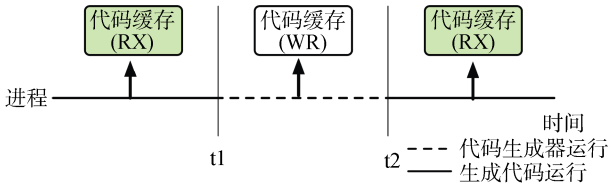

图 4 基于权限转换的 W⊕X 方法[18]

与上述时域的数据执行保护不同, 另一种空域的数据执行保护方法将动态代码生成中生成代码和执行代码两个功能放在不同的进程中实现, 通过共享内存等内存映射机制共享代码缓存, 并设置不同的访问权限。我们称这种方法为基于权限分离的数

据执行保护。

Niu 等人在 RockJIT[9]的实现中利用虚拟内存和沙盒机制将代码缓存区和 RockJIT 中影子代码堆(shadow code heap)映射到相同的物理地址并设置不同的权限, 如图 5 所示。用沙盒机制将原代码缓存区设置为不可写可执行, 将RockJIT中的影子代码堆设置为可写不可执行。即时编译器生成的代码经过RockJIT 的安全验证之后写入影子代码堆。而执行生成代码时依然访问原代码缓存。这使得即时编译器在向代码缓存区写入代码时, 只能通过RockJIT写入影子代码堆且必须通过 RockJIT 的控制流完整性检测。即便攻击者能够在即时编译器的沙盒中任意写, 也不能够使恶意代码通过RockJIT的验证, 所以无法执行 Shellcode。

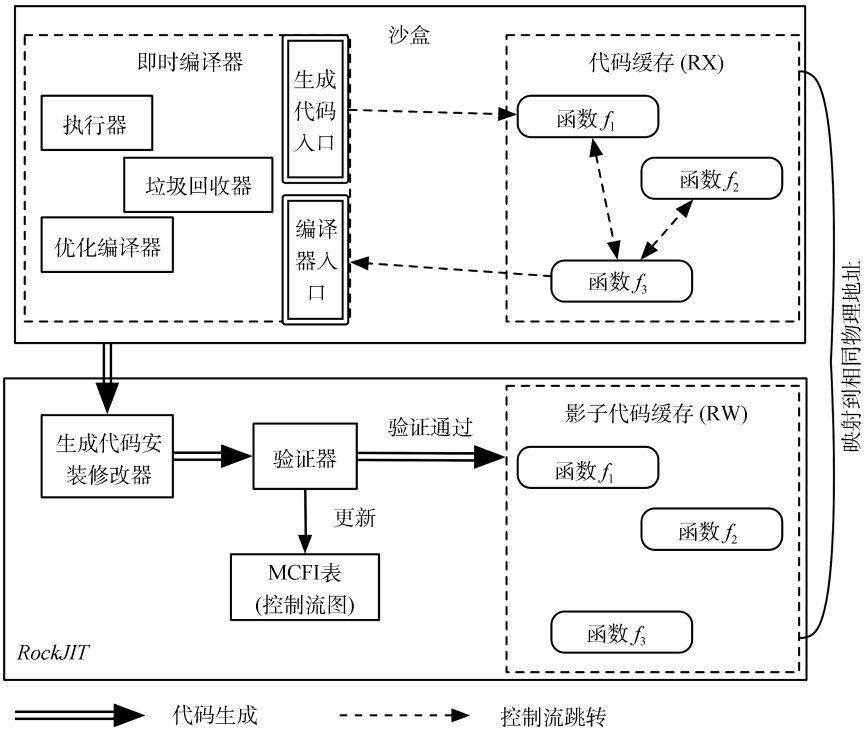

图 5 RockJIT 架构[22]

另一种基于权限分离数据执行保护的方法是 Jaurnig 等人[20]提出的 Lobotomy。该方法在 Firefox TraceMonkey 即时编译器中应用, 用于防御 JIT 喷射攻击。Lobotomy 将 JIT 引擎分为编译器和执行器两个进程, 并共享代码缓存区。编译进程对缓存区可写不可执行, 执行进程对缓存区可写可执行。通过上述权限分离, 使得攻击者无法利用编译进程中的控制流劫持漏洞执行注入的代码。

Song 等人在文献[18]中指出在多线程环境下[53], 即便部署了基于权限转换的 $W \oplus X$[19], 攻击者依然可以进行基于竞争条件(race condition)的攻击, 在攻击窗口内向代码缓存区注入 Shellcode。为解决基于权限转换的数据执行保护在多线程环境下易受竞争情形攻击的弱点, Song 等人提出了一种广泛适用于即时编译和动态二进制转换的安全动态代码生成框架 SDCG(Secure dynamic code generation), 可以为可信的软件动态翻译器提供健壮的数据执行保护。竞争条件攻击利用动态翻译器为一个线程生成代码时将代码缓存区设置为可写的特点, 利用另一线程向代码缓存区注入 Shellcode。如图 6 所示, 线程 A 在第①次访问代码缓存时没有修改代码的权限, 访问②表示代码生成器正在为线程 A 生成代码, 访问③表示线程 B 也具有了对代码缓存的写权限所以可以注入 Shellcode, 发起访问④时线程 A 已经完成所需代码的生成, 故所有线程不再具有对代码缓存的写权限, 因此代码注入攻击的时间窗口长度是 t2-t1。这种攻击利用了内存访问权限的粒度只到进程级别, 同一进程中的线程具有相同的访问权限。当一个线程生成代码时, 其他线程都会获得写代码缓存的权限, 这就方便了代码注入攻击。针对这类攻击, SDCG 的核心是一种通过共享内存将相同的内存映射到不同的进程, 并为不同进程设置不同的访问权限的技术。将属于可信计算基(TCB)的软件动态翻译器放到一个专用进程中, 其他进程通过远程过程调用(Remote procedure call)使用其动态生成代码的功能。在软件动态翻译器进程中, 代码缓存区被设置为可

写不可执行, 而在其他进程中代码缓存区被设置为不可写可执行。上述内存访问权限设置通过基于代理的沙盒架构(delegation-based sandbox architecture)[54]拦截所有关于虚拟内存管理的系统调用实现以确保权限设置不被篡改。

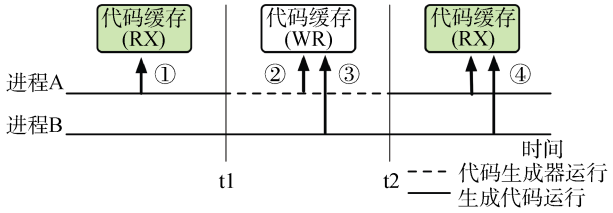

图 6 使用双线程进行竞争条件攻击[18]

5.1.2 控制流完整性

控制流完整性保护[10]已经历了 10 年的发展, 被证明是一种防御控制流劫持攻击的有效手段并开始被工业界采用, Visual Studio 2015 中已经支持将控制流保护信息写入 PE 文件格式中, 但是传统控制流完整性保护方法并不适用于动态代码生成。为动态代码生成程序实施控制流完整性保护除了需要为程序本身插装外, 还需要保护代码缓存区中动态生成的代码, 其难点如图 7 所示。传统控制流完整性采用的静态分析和插装的工作方式可以保护动态代码翻译器的控制流。使得被保护程序的控制流跳转时检查控制流跳转目标, 确保控制流跳转目标在通过静态分析确定的控制流图中。但传统控制流完整性对于动态生成的可执行代码则无能为力。动态代码生成需要在运行时动态生成或修改代码, 这样的特性与传统的控制流完整性的静态分析与改写的工作模式矛盾, 因此控制流完整性安全策略不能直接用于保护 JIT 编译等动态代码生成技术。

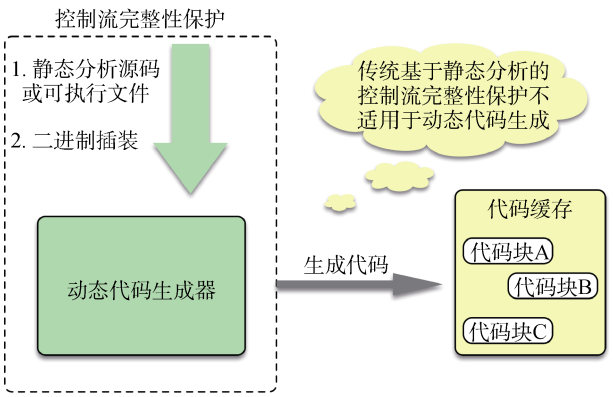

图 7 控制流完整性保护动态生成代码的难点

为了弥补传统控制流完整性方法的不足, Ansel 等人在文献[21]中提出了 NaCl-JIT, 一种用基于本地客户端沙盒(Native Client Sandbox[55], 简称 NaCl)的保护动态代码生成的方案。NaCl 利用基于软件的错误分离实现了沙盒, 它通过静态分析和插装软件护卫(software guards)限制对内存的访问。为保护动态代码生成, NaCl-JIT 扩展了 NaCl 的编程接口以接管动态添加、修改和删除代码等操作。在动态生成代码时, NaCl-JIT 检查生成代码满足以下安全性质: 生成代码的目标地址在 NaCl 的可执行内存区域内; 使用标准的 NaCl 验证器对生成代码进行验证, 生成代码的目标地址必须是没有被使用过的保留内存空间; 并限制控制流的跳转目标必须是对齐的地址(32 字节对齐)。这实际上也通过限制直接与间接分支跳转目标, 实现了对即时编译器和代码缓存的一种粗粒度控制流完整性保护。最近的研究证明粗粒度的控制流完整性策略不能有效保护程序的控制流, 精心构造的 ROP 攻击[56,57]可以绕过粗粒度的控制流完整性保护, NaCl-JIT 实现的控制流完整性难以防御这类攻击。而且 NaCl-JIT 由于大量使用 NOP 填充以对齐指令, 造成了较高的运行时开销。

为了弥补传统控制流完整性方案不能应用于动态生成代码的不足, Niu 等人在文献[22]中提出了如图 5 所示的加固即时编译的方案: RockJIT, 一种将控制流完整性策略应用于即时编译的方法。RockJIT 提出了一种为即时编译器进行安全加固的架构。为即时编译器实施基于 MCFI[42]的细粒度的控制流完整性保护并在运行时用粗粒度的控制流完整性保护动态生成的代码。通过基于软件错误分离的沙盒限制即时编译器的系统调用, 并确保动态生成代码在正常调用不含敏感系统调用, 因此为 JIT 生成代码执行粗粒度的控制流完整性就已足够。RockJIT 通过静态分析即时编译器的源代码以产生细粒度的 CFG, 为即时编译器实施细粒度的控制流完整性保护。为用控制流完整性保护动态代码生成, 在向代码缓存区添加新的 JIT 生成代码、修改已有的 JIT 生成代码和删除 JIT 生成代码时需要通过 RockJIT 的验证以更新控制流策略。为配合 RockJIT, 对即时编译器源代码进行了修改并将其和代码缓存放在沙盒之中以确保即时编译器不再具有对 JIT 生成代码堆的写权限, 即时编译器进行代码生成、修改、删除时需要通过 RockJIT 中验证器的控制流完整性保护验证, 才能被写到 RockJIT 内存中的影子代码堆, 从而防止代码注入攻击。

5.2 闪避防御

闪避防御(Moving Target Defense)[44]是一种新的防御思想。其核心思想与拟态安全防御(Mimic secu-

rity defense)[33]类似, 通过对系统配置的灵活变化, 变被动为主动, 向攻击者展现一个不断变化的不可预测的攻击面, 大幅提高攻击者的利用难度。从攻击者的角度, 动态生成代码技术的代码缓存区是一个倍受关注的攻击点。其可写可执行的权限设置使其成为恶意代码注入的入口。动态代码生成器内在逻辑的确定性使得攻击者可以通过操纵输入程序获得其需要的 Shellcode。闪避防御技术通过移动代码段和各种随机化方法为动态生成代码添加不确定性, 将攻击成功变为一个概率极小的事件。

闪避防御技术的有效性需要针对一个既定的威胁模型进行衡量。现有的面向动态代码生成的闪避防御手段通常针对如下的威胁模型: 程序中存在控制流劫持漏洞, 攻击者通过控制输入代码产生所需 Shellcode 或者工具代码片段。生成代码多样化通过为生成代码添加了不确定性防御上述威胁。

5.2.1 生成代码多样化

代码重用攻击和代码注入攻击依赖于生成代码的位置和内容具有一定可预测性。研究者提出一系列打破生成代码可预测性的多样化方法[23-26]。生成代码多样化产生与原生成代码语义相同但形态不同的代码, 可以使得生成代码的位置和内容对于攻击者而言不可预测。从而防止攻击者通过立即数引入 Shellcode 或工具代码片段。

为打破生成代码的可预测性, 北京大学计算机科学技术研究所的韦韬等人在文献[24]中提出一种生成代码多样化技术: INSeRT。针对 X86 指令格式提出了立即数随机化(Immediate randomization)、寄存器随机化(Register randomization)、位移随机化(Displacement randomization)等技术。立即数随机化将立即数与一个随机数异或使得攻击者不再能控制立即数的值。如指令“*mov eax*, *imm*32”就被变换为“*mov eax*, (*imm*32^*rand_seed*); *xor eax*, *rand_seed*”。寄存器随机化将使用各个寄存器的顺序随机化。位移随机化将函数的参数和与局部变量的位置打乱以随机化指令中的位移域。与上述方法类似, 陈平等人在 RIM[23]和 JITsafe[26]中也使用了立即数随机化和寄存器随机化的方法。

为应对无法获得软件动态翻译器的源代码的情况, Homescu 等人在文献[25]提出了一种无需修改动态代码生成器的生成代码多样化方法 Librando 以加固即时编译。Librando 设置代码缓存区为不可执行并截取所有在代码缓存区中开辟可执行区域的调用, 将动态生成代码反编译为控制流图, 对其中每个基本块进行多样化后写入另一块可执行内存区域。所有跳转到原生成代码的分支都被重定向到多样化后的生成代码。其多样化技术包括了空指令插入(NOP insertion)和常量盲化(Constant blinding)。空指令插入技术在生成代码中随机插入各种不影响代码语义的空指令, 从而使得生成代码的位置和任意两段生成代码的相对距离都变得不确定, 算法 1 是一种空指令插入算法。常量盲化将生成代码中的立即数加密, 使得攻击者不再能够控制生成代码中立即数的值。如图 8 所示, 常量盲化是一种在生成代码中隐藏输入常量的方法, 通过从立即数中减去一个随机数将每一个产生的立即数加密, 并利用 LEA 指令在运行时对立即数进行解密。

算法 1.一种空指令插入算法[25].

输入: 基本块 *B*, 空指令插入概率 *p*

输出: 被插入空指令的基本块 *B*

```
FOR each instruction i in B DO
v = random(0,1)
IF v<p THEN
    insert( i, randomNOP())
END
END
```

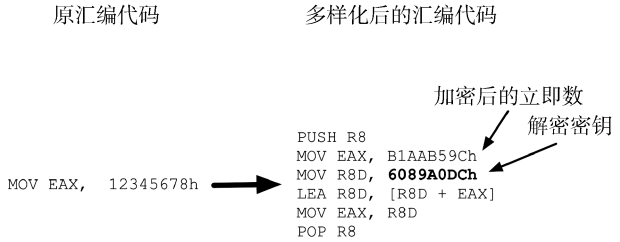

图 8 常量盲化实例[25]

6 安全性度量

Merkov 等人在文献[58]中指出, 没有防御手段能实现绝对的安全, 攻击者在有足够时间、工具、技能和动力的情况下可以攻破任何系统。因此防御手段的目的是提高攻击者的攻击成本, 同时将在系统中部署防御的开销保持在应用场景可接受的范围内。研究者提出了许多加固动态代码生成技术的防御手段, 但是并没有统一的衡量防御效果的标准, 我们结合动态代码生成技术的特点, 首次提出衡量动态代码生成安全性的模型, 并从防御效果和防御成本两方面衡量动态生成代码系统的安全性。

6.1 防御效果

防御效果衡量防御技术提高攻击难度的程度, 数据执行保护和控制流完整性等强制性防御手段旨在防御攻击中的某个必要步骤, 而生成代码多样化等闪避防御旨在提供一个在攻击者视角中变幻莫测

的攻击面。

6.1.1 衡量强制性防御效果

强制性防御使特定攻击方式或步骤无效, 例如: 细粒度的控制流完整性保护可以防止控制流跳转到控制流图外的地址从而防止 ROP 攻击[22]; 基于权限转换的数据执行保护缩减了攻击者注入和执行恶意代码的攻击窗口, 所以能够提高代码注入攻击的难度[19], 迫使攻击者利用多线程环境下的竞争条件攻击[18]。发现一类新的攻击的难度大于应用已知攻击。迫使攻击者发现一类新的攻击方式能大幅提高攻击成本。我们衡量了强制性防御手段对代码注入攻击和代码重用攻击的影响, 表 1 对面向动态生成代码的典型强制性防御技术进行了对比分析。根据防范代码注入和代码重用两种攻击的机制, 分析其存在的主要威胁向量。

防御代码重用攻击的核心手段是保护控制流完整性。衡量控制流完整性保护有效性的两个指标是其移除 ROP 工具代码片段的数量和移除间接分支(Indirect branch)的数量。利用上述指标, Niu 等人在对 RockJIT[22]的安全性进行了如下衡量: 1)利用工具代码片段查找工具在用细粒度控制流完整性保护加固后的 V8 即时编译器中查找工具代码片段。其结果显示 RockJIT 可以移除 V8 即时编译器中 98.5%的 ROP-工具代码片段; 2)统计 RockJIT 为 V8 即时编译器生成的控制流图中允许的间接分支跳转的数量, 并与 NaCl-JIT 为 V8 进行粗粒度的控制流完整性保护后所允许的间接分支跳转的数量进行对比, 结果表明, RockJIT 比 NaCl-JIT 多移除了 99.97%的间接分支跳转。

6.1.2 衡量闪避防御效果

闪避防御是一种变化攻击面的防御手段。最近研究者提出了一系列评价闪避防御效果的方法[59-61]。Hobson 等人在文献[59]中提出了衡量闪避防御有效性的三个维度: 覆盖面(coverage)、不可预测性(unpredictability)和及时性(timeliness)。由于三者中任何一方面的不足都可能为攻击者提供一部分确定的攻击面, 因此闪避防御手段要满足以下三个条件才能显著提高攻击成本: 1)覆盖所有可能被利用的攻击面; 2)具有足够高的熵值; 3)能及时变换不给攻击者留下足够长的攻击窗口以进行基于信息泄露的攻击。我们参考文献[59]的评价方式, 在表 2 对现有的针对动态代码生成的闪避防御有效性进行了衡量, 从以上三个方面列出了其不足之处。

对于生成代码多样化这类闪避防御, 其覆盖面只包含生成代码, 并未提高对系统其他部分的攻击难度。Athanasakis 等人在文献[15]中认为现有代码多样化技术的最大缺陷是及时性的不足: 利用了信息泄露漏洞的代码重用攻击可以在运行时读取生成代码并利用其构建 ROP 攻击, 从而完全绕过代码多样化防御。Athanasakis 用上述方法成功攻击了 Firefox 和 Internet Explorer 两款常用浏览器。

表 1 强制性防御有效性衡量

防御方法	防范代码注入攻击的机制		防范动态生成代码重用攻击的机制	主要威胁向量
	防范 JIT 喷射攻击的机制	防范代码缓存覆盖攻击的机制		
NaCl-JIT	扩展的 Native Client 沙盒		粗粒度的控制流完整性	代码重用
Lobotomy	分离编译器和执行器	无	缩减攻击面	代码缓存覆盖、代码重用
RockJIT	内存映射和沙盒		粗细粒度控制流完整性的组合	无
JIT-Defender	基于权限转换的数据执行保护, 缩减攻击窗口		缩减攻击窗口	代码缓存覆盖
JIT-safe	基于权限转换的数据执行保护和数据、代码分离		缩减攻击窗口	代码缓存覆盖
SDCG	无	基于进程间内存共享的数据执行保护	缩减攻击面	JIT 喷射、代码重用

表 2 闪避防御有效性的衡量

防御方法	覆盖面(Coverage)		不可预测性(Unpredictability)		及时性(Timeliness)
	软件动态翻译器	代码缓存	优点	不足	
地址空间随机化	是	是	将生成代码段的起始地址随机化	熵较小, 易受暴力猜解和 JIT 喷射攻击	在程序加载时随机化, 但地址泄露发生在运行时
INSeRT	否	是	对整个指令集空间进行随机化	无明显不足	在代码生成时进行随机化, 代码生成后可能泄露
JITsafe	否	是	对指令进行随机化	无明显不足	在代码生成时进行随机化, 代码生成后可能泄露
Librando	否	是	采用随机 NOP 指令插入和常量盲化	对小于两字节的常量未进行盲化	在代码生成时进行随机化, 代码生成后可能泄露

6.2 防御成本

除了提高攻击成本的能力, 衡量面向动态生成代码的防御手段的实用性还需要考虑系统性能损失、实现复杂度、是否需要源代码等因素。表 3 从保护对象、系统性能损失、代码修改量和是否需要源代码四个方面对研究者提出的各种防御技术的防御成本进行对比分析。

系统性能损失是动态代码生成场景下最重要的防御成本因素。动态代码生成系统的可用性不应被防御技术损害。特别是更快、更高效是在浏览器, Flash Player 等程序中应用动态代码生成的主要目的, 因此动态代码生成安全加固措施应当具有较低的运行时开销。如果运行时时间开销超过 5%~10%, 工业界就不愿牺牲性能而采取这样的防御措施[6]。

代码修改行数体现了防御机制的复杂性。复杂的安全机制需要投入更多资源实现, 同时复杂性本身也会带来新的缺陷和漏洞。黑盒的防御方式(如 Librando)不需要对保护对象的源代码进行修改。

源代码需求对部署防御的可行性有一定影响: 1)依赖源代码的防御手段在不开源的动态代码生成器上就不适用; 2)在新的威胁出现时, 不需要修改程序源代码的防御措施(例如 Librando)可以迅速给系统打补丁而无需等待厂商推出安全补丁。

7 技术展望

攻击动态生成代码系统的难度随着防御机制发展不断增加, 需要针对系统防御机制研究攻击技术。

防御技术的有效性和开销依然存在矛盾(例如: 细粒度的控制流完整性具有较高的运行时开销), 所以有必要研究运行时开销更少且不削弱防御有效性的防御技术。

与细粒度的控制流完整性保护相比, Kuznetsov 在 2014 年提出的代码指针完整性(Code pointer integrity, 简称 CPI)[62]是一种同样有效, 并且运行时开销更小的强制性防御技术, 其实现已经开源。但将 CPI 应用于动态代码生成还存在着很大的挑战, 因此仍然需要对适用于动态代码生成的代码指针完整性进行深入的研究。

表 3 防御成本衡量

防御方法	保护对象	系统性能损失/Benchmark	代码修改行数	源代码
INSeRT	V8 JavaScript engine	小于 5%/SunSpider 0.9.1 JS benchmark	N/A	需要
NaCl-JIT	V8 JavaScript engine/ MonoCLR	28%(x86-32) 51%(x86-64)/ V8 benchmark suite	超过 5000 行	需要
JIT-Defender	Tamarin Flash/ V8 JavaScript engine/ Safari JavaScript	均小于 1%	N/A	需要
RIM	Tamarin Flash engine	小于 1%/ Tamarin Flash benchmark	N/A	需要
Librando	V8 JavaScript engine/ Java HotSpot	265.8%/ V8 benchmark suite	0	不需要
Lobotomy	Firefox Tracemonkey JIT engine	27%/ N/A	N/A	需要
RockJIT	V8 JavaScript engine	14.6%/ Octane 2 JS benchmark	约 800 行	需要
SDCG	V8 JavaScript engine	6.9%(32-bit) 5.65%(64-bit)/ V8 benchmark suite	约 2500 行	需要
	Strata	1.64%/ SPEC CINT 2006	约 1000 行	需要

闪避防御及其理论仍然处于起步阶段, 依然有许多问题等待进一步研究解决。例如: 面向动态生成代码的多样性防御技术不能有效防御内存泄露攻击[38,40]尽管最近研究者们提出了一些应对信息泄露攻击的防御技术[63,64], 但还并没有被工业界采用。而且, 将此类技术应用于动态代码生成存在着新的挑战。

对于动态代码生成系统源代码不可得的情况, 依然缺少低开销的黑盒防御技术。因此有必要对高效的黑盒防御技术展开研究。

没有一种防御手段能够一劳永逸地解决所有安全问题。为保护动态代码生成, 需要将不同的防御手段结合起来。组合防御的优点在于防御手段可以相互补充彼此的弱点。强制性防御和闪避防御的巧妙组合将可能增加系统的安全性[63,64]。理想的分层布置防御应当使得攻击成本能够随防御层数增加而指数性增长[65], 而实际中防御手段的组合只能线性提高攻击难度, 因此更有效的组合防御方法还有待研究。

8 总结

动态代码生成在运行时生成、修改可执行代码的特性作为提高动态语言执行速度和实现虚拟化的关键技术, 也可以被攻击者利用实现任意代码执行或高级可持续威胁。如今动态代码生成的应用场景几乎无处不在, 其安全性问题尤为重要。本文对面向动态代码生成的攻击和防御技术进行了综述。介绍了动态生成代码的数据平面和代码平面耦合导致的安全问题, 针对动态代码生成的代码注入攻击和代码重用攻击的特点, 分析了对应防御方法的实现方法, 并对防御效果进行了衡量。

目前, 工业界对于动态生成代码防御并无成熟的解决方案, 其攻防关键技术(例如: 高级 ROP 攻击与防御、软件多样化和闪避防御)的研究状态还处在“魔高一尺, 道高一丈”的对抗发展阶段。安全性和防御成本依然是需要权衡的矛盾, 有效且低成本的防御技术是研究的重点。

致 谢

在此向对本文的工作给予建议的龚晓锐老师和给我们提出建议的评审专家 表示感谢。

参考文献

[1] “V8 JavaScript Engine,” Google, https://developers.google.com/v8/design.

[2] “SpiderMonkey 1.8.8,”Mozilla, 2014.

[3] “Java Hotspot,” Oracle, http://www.oracle.com/technetwork/java/whitepaper-135217.html.

[4] Facebook. HHVM. http://hhvm.com/.

[5] F. Bellard, “QEMU, a Fast and Portable Dynamic Translator,” *in Proceedings of the annual conference on USENIX Annual Technical Conference(ATEC’05)*, pp. 41–46, 2005.

[6] L. Szekeres, M. Payer, T. Wei and D. Song, “SoK: Eternal war in memory,” in *Proc. IEEE Symp. Security and Privacy(SP’13)*, pp. 48-62, 2013.

[7] One A. “Smashing the stack for fun and profit,” *Phrack magazine*, vol. 7, no. 49,pp. 14-16, 1996.

[8] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, “Return-oriented programming: Systems, languages, and applications,” In *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 1,pp. 2, 2012.

[9] T. Bletsch, “Code-reuse attacks: new frontiers and defenses [Ph.D. dissertation],” *North Carolina State University*, 2011.

[10] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, “Control-flow integrity principles, implementations, and applications,” In *ACM Transactions on Information System Security (TISSEC)*, vol. 13, no. 4, 2009.

[11] P. Larsen, A. Homescu, S. Brunthaler and M. Franz, “SoK: Automated Software Diversity,” In *Proc. IEEE Symp. Security and Privacy(SP’14)*, pp. 276-291, 2014.

[12] D. Blazakis, “Interpreter exploitation,” In *Proceedings of the 4th USENIX conference on Offensive technologies*, 2010.

[13] C. Rohlf and Y. Ivnitskiy. “Attacking clientside JIT compilers,” *Black Hat USA*, 2011.

[14] P. Pie, “chrome on Android exploit writeup,” *Mobile Pwn2Own*, Autumn, 2013.

[15] Athanasakis M, Athanasopoulos E, Polychronakis M, et al. “The Devil is in the Constants: Bypassing Defenses in Browser JIT Engines,” In *Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium*. 2015.

[16] “Flash JIT – Spraying info leak gadgets,” F. J. Serna, http:// zhodiac.hispahack.com/my-stuff/security/Flash_Jit_InfoLeak_Gadgets.pdf, July, 2013.

[17] “Writing JIT-spray shellcode for fun and profit,” A. Sintsov, https://packetstormsecurity.com/files/download/86975/Writing-JIT-Spray-Shellcode.pdf, 2010.

[18] C. Song, C. Zhang, T. Wang, W. Lee and D. Melski, “Exploiting and Protecting Dynamic Code Generation,” In *Proceedings of the 2015 Network and Distributed System Security (NDSS) Symposium*, 2015.

[19] P. Chen, Y. Fang, B. Mao and X. Li, “JITDefender: A defense against JIT spraying attacks.” In *Future Challenges in Security and Privacy for Academia and Industry*, Springer Berlin Heidelberg, pp. 142-153, 2011.

[20] M. Jauernig, M. Neugschwandtner, C. Platzer and P.M. Comparetti, “Lobotomy: An Architecture for JIT Spraying Mitigation,” In *Proceedings of the Ninth International Conference on Availability, Reliability and Security(ARES)*, pp. 50-58, 2014.

[21] J. Ansel, P. Marchenko, Ú Erlingsson, E. Taylor, B. Chen, D. Sehr, C.L. Biffle and B. Yee, “Language-independent sandboxing of just-in-time compilation and self-modifying code,” In *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 355-366, 2011.

[22] B. Niu, G. Tan, “RockJIT: Securing just-in-time compilation using modular control-flow integrity,” In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security(CCS’14)*, pp. 1317-1328, 2014.

[23] R. Wu, P. Chen, B. Mao and L. Xie. “RIM: A Method to Defend from JIT Spraying Attack,” In *Proceedings of the Seventh International Conference on Availability, Reliability and Security(ARES)*, pp. 143-148, 2012.

[24] T. Wei, T. Wang, L. Duan and J. Luo. “INSeRT: Protect dynamic code generation against spraying,” In *Proceedings of International*

Conference on Information Science and Technology (ICIST), pp. 323-328, 2011.

[25] A. Homescu, S. Brunthaler, P. Larsen and M. Franz, "librando: Transparent code randomization for just-in-time compilers," In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security(CCS'13)*, pp. 993-1004, 2013.

[26] P. Chen, R. Wu, B. Mao, "JITSafe: a framework against Just-in-time spraying attacks," In *IET Information Security*, vol.7 no.4 pp. 283-292, 2013.

[27] W. De Groef, N. Nikiforakis, Y. Younan, and F. Piessens. "Jitsec: Just-in-time security for code injection attacks," In *Proceedings of the Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 1-15, 2010.

[28] C. Zhang, M. Niknami, K. Chen, C. Song, Z. C and D. Song, "JITScope: Protecting web users from control-flow hijacking attacks," In *Proceedings of 2015 IEEE Conference on Computer Communications (INFOCOM),*pp. 567-575, 2015.

[29] J. Li, X. Ma and C. Zhu, "Dynamic Binary Translation and Optimization," Journal of Computer Research and Development, vol. 44, no. 1, pp.161-168, 2007.
(李剑慧, 马湘宁, 朱传琪, "动态二进制翻译与优化技术研究", 计算机研究与发展, 2007, 44(1): 161-168.)

[30] T. Bai, X. Feng, C, Wu and Z. Zhang, "Optimizing Dynamic Binary Translator in DigitalBridge," *Journal of Computer Engineering*, vol. 31, no.10, pp. 103-105, 2005.
(白童心, 冯晓兵, 武成岗, 张兆庆, "优化动态二进制翻译器DigitalBridge",计算机工程, 2005, 31(10).)

[31] C.K. Luk, R. Cohn, R. Muth, H. Ratil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," In *ACM Sigplan Notices*, vol. 40, no. 6, pp. 190-200, 2005.

[32] E.G. Barrantes, D.H. Ackley, T.S. Palmer, D. Stefanovic and D.D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks," In *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 281-289, 2003.

[33] J. Wu, "Meaning and Vision of Mimic Computing and Mimic Security Defense," in Telecommunications Science, vol. 1, no.7, pp. 1-7, 2014.
(邬江兴, "专题导读--拟态计算与拟态安全防御的原意和愿景", 电信科学, 2014, 30(7): 1-7.)

[34] R. Skowyra, K. Casteel, H. Okhravi, N. Zeldovich and W. Streilein, "Systematic Analysis of Defenses against Return-Oriented Programming," In *Research in Attacks, Intrusions, and Defenses*, *Springer Berlin Heidelberg*, pp. 82-102, 2013.

[35] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu and D. Boneh. "On the effectiveness of address-space randomization," In *Proceedings of the 11th ACM conference on Computer and communications security(CCS'04)*, pp. 298-307, 2004.

[36] Strackx R, Younan Y, Philippaerts P, F. Piessens, S. Lachmund and T. Walter. "Breaking the memory secrecy assumption," In *Proceedings of the Second European Workshop on System Security(EUROSEC'09)*, pp. 1-8, 2009.

[37] G.F. Roglia, L. Martignoni, R. Paleari and D. Bruschi. "Surgically returning to randomized lib (c)," in *Proceedings of the 25th Annual Computer Security Applications Conference(ACSAC'09)*, pp. 60-69, 2009.

[38] A. Bittau, A. Belay, A. Mashtizadeh and D. Boneh, "Hacking blind," In *Proceedings of the 35th IEEE Symposium on Security and Privacy(SP'14)*, pp. 227-242, 2014.

[39] F.J. Serna, "The info leak era on software exploitation," *Black Hat USA*, 2012.

[40] K.Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen and A. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," In *Proceedings of the 34th IEEE Symposium on Security and Privacy(SP'13)*, pp. 574-588, 2013.

[41] C. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres, S. McCamant, D. Song and W. Zou, "Practical control flow integrity and randomization for binary executables," In *Proceedings of the 34th IEEE Symposium on Security and Privacy(SP'13)*, pp. 559-573, 2013.

[42] B. Niu, G. Tan, "Modular control-flow integrity," In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI'14)*, pp. 577-587, 2014.

[43] M. Zhang M, R. Sekar, "Control Flow Integrity for COTS Binaries," In *Proceeding of the 22nd USENIX Security Symposium(SEC'13)*, pp. 337-352, 2013.

[44] S. Jajodia, A.K. Ghosh, V. Swarup, C. Wang and X.S. Wang, "Moving Target Defense," *Springer*, 2011.

[45] "Safari JS JITed Shellcode," A. Sintsov, http://www.exploit-db.com/exploits/14221/ , 2010.

[46] M. Egele, P. Wurzinger, C. Kruegel and E. Kirda, "Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks," In *Detection of Intrusions and Malware, and Vulnerability Assessment. Springer Berlin Heidelberg*, pp. 88-106, 2009.

[47] C. Rohlf, Y. Ivnitskiy, "The security challenges of client-side just-in-time engines," In *IEEE Security & Privacy*, vol. 10, no. 2, pp. 84-86, 2012.

[48] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti and E. Kirda, "G-Free: defeating return-oriented programming through gadget-less binaries," In *Proceedings of the 26th Annual Computer Security Applications Conference(ACSAC'10)*, pp. 49-58, 2010.

[49] C. Zhang, T. Wang, T. Wei, Y. Chen and W. Zou, "IntPatch: Auto-

matically fix integer-overflow-to-buffer- overflow vulnerability at compile-time," In *Proceedings of the 15th European conference on Research in computer security(ESORICS'10)*, pp. 71-86, 2010.

[50] J. Ye, C. Zhang and X. Han, "POSTER: UAFChecker: Scalable Static Detection of Use-After-Free Vulnerabilities," In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security(CCS'14)*, pp. 1529-1531, 2014.

[51] M. Sutton, A. Greene and P. Amini, "Fuzzing: brute force vulnerability discovery," *Pearson Education*, 2007.

[52] T. Wang, T. Wei, G. Gu and W. Zou, "TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection," In *Proceedings of the 31th IEEE Symposium on Security and Privacy(SP'10)*, pp. 497-512, 2010.

[53] "Web Workers," W3C, http://www.w3.org/TR/workers/, 2012.

[54] T. Garfinkel, B. Pfaff, M. Rosenblum, "Ostia: A Delegating Architecture for Secure System Call Interposition," In *Proceedings of the 2004 Network and Distributed System Security (NDSS) Symposium*, 2004.

[55] B. Yee, D. Sehr, G. Dardyk, J.B. Chen, R. Myth, T. Ormandy, S. Okasaka, N. Narula and N. Fullagar. "Native client: A sandbox for portable, untrusted x86 native code," In *Proceedings of the 30th IEEE Symposium on Security and Privacy(SP'09)*, pp. 79-93, 2009.

[56] N. Carlini, D. Wagner, "Rop is still dangerous: Breaking modern defenses," In *Proceeding of the 23rd USENIX Security Symposium(SEC'14)*, pp. 385-399, 2014.

[57] E. Göktas, E. Athanasopoulos, H. Bos and G. Portokalidis, "Out of control: Overcoming control-flow integrity," In *Proceedings of the 35th IEEE Symposium on Security and Privacy(SP'14)*, pp. 575-589, 2014.

[58] Stamp M, "Information security: principles and practice," *John Wiley & Sons*, 2011.

[59] T. Hobson, H. Okhravi, D. Bigelow, R. Rudd and W. Streilein, "On the Challenges of Effective Movement," In *Proceedings of the First ACM Workshop on Moving Target Defense(MTD 2014)*, pp. 41-50, 2014.

[60] J. Xu, P. Guo, M. Zhao, R.F. Erbacher, M. Zhu and P. Liu, "Comparing Different Moving Target Defense Techniques," In *Proceedings of the First ACM Workshop on Moving Target Defense(MTD 2014)*, pp. 97-107, 2014.

[61] G. Cybenko and J. Hughes, "No free lunch in cyber security," In *Proceedings of the First ACM Workshop on Moving Target Defense(MTD 2014)*, pp. 1-12, 2014.

[62] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar and D. Song, "Code Pointer Integrity," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation(OSDI'14)*, pp. 147-163, 2014.

[63] L. Davi, C. Liebchen, A. Sadeghi, K.Z. Snow and F. Monrose, "Isomeron: Code randomization resilient to (just-in-time) return-oriented programming," In *Proceedings of the 22nd Network and Distributed System Security (NDSS) Symposium*. 2015.

[64] V. Mohan and P. Larsen, S. Brunthaler, K. Hamlen and M. Franz, "Opaque control-flow integrity," In *Proceedings of the 22nd Network and Distributed System Security (NDSS) Symposium*, 2015.

[65] S.M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," In *IEEE Security & Privacy*, vol. 4, no. 4, pp. 96-96, 2006.

[66] "The Enhanced Mitigation Experience Toolkit," Microsoft, https://support.microsoft.com/en-us/kb/2458544.

[67] "grsecurity," grsecurity, http://grsecurity.net/features.php.

[68] "Usage of JavaScript for websites," W3techs, http://w3techs.com/technologies/details/cp-JavaScript/all/all,2014.

吴炜 2014 年在中国科学技术大学信息安全专业获得学士学位。现在中国科学院信息工程研究所攻读博士学位。研究领域为网络与软件安全。研究兴趣包括: 漏洞挖掘与利用、程序分析及网络对抗技术。Email: wuwei@iie.ac.cn

邹维 研究员、博士生导师。现任中国科学院信息工程研究所副所长。中国科学院"百人计划"人选, 中国计算机学会优秀博士论文指导教师。研究领域包括网络与软件安全、攻防对抗理论与技术等。Email: zouwei@iie.ac.cn

霍玮 博士, 副研究员。2010 年在中国科学院计算技术研究所获得博士学位。主要研究方向为软件漏洞挖掘和安全评测、基于大数据的软件安全分析、智能终端系统及应用安全分析等。Email: huowei@iie.ac.cn