基于自适应模糊测试的 IaaS 层漏洞挖掘方法

沙乐天[12] 肖甫*[12] 杨红柯[3] 王汝传[12]

（1. 南京邮电大学计算机学院，江苏南京，210023；2. 江苏省无线传感网高技术研究重点实验室，江苏南京，210023；3. 华为企业通信技术有限公司杭州研究所，浙江杭州，310052）

摘要云计算在为人们日常生活提供极大便利的同时，也带来了较大安全威胁。近年来云平台 IaaS 层虚拟化机制的漏洞层出不穷，如何有效挖掘虚拟化实现过程中的拒绝服务及逃逸漏洞是当前的研究难点。本文分析已知虚拟化平台的相关漏洞，抽取并推演目标数据集合，设计并实现了一种随机化的模糊测试方法，进一步，基于灰度马尔科夫模型设计了一种自动化预测方法，实时监督并调整模糊测试的方向，实现面向虚拟化平台的自适应模糊测试目的。最终设计并实现了原型系统 VirtualFuzz，实验数据表明：所提方法可有效检测虚拟化平台中的拒绝服务及逃逸漏洞，共得到 24 个漏洞测试用例，其中验证了 18 个已知漏洞，挖掘得到了 6 个未知漏洞，且已有 3 个漏洞获得 CVE 授权；同时通过与其他模糊测试工具的对比突出了原型系统的性能优化效果。

关键词自适应；模糊测试；灰度马尔科夫

中图法分类号： TP393 文献标识码：A

A Vulnerability Discovery Method for Virtualization in IaaS Based on Self-adapting Fuzzing Test

Letian Sha[12], Xiao Fu*[12], Hongke Yang[3], Ru-chuan Wang[12]

(1. College of Computer, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu, China; 2. Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks, Nanjing, Jiangsu, China; 3. Hangzhou Technology Institute of HUAWEI Company, Hangzhou, 310052, China)

Abstract: It has provided large convenience for data lifetime of people by cloud computing, however, huge security threatens has been introduced via related technology. Recently more and more vulnerabilities have been discovered for virtualization in IaaS of cloud platform, it can be viewed as a difficult problem to discover DDoS and Escape vulnerabilities in virtualization mechanism. In this paper, some known bugs are analyzed for related platforms, target test case sets are extracted and extended, and randomized fuzzing test is designed and accomplished. Finally, an automatic prediction is proposed based on gray Markova model, via which the direction of fuzzing test can be supervised and adjusted in real time, and self-adapting fuzzing test can be achieved for virtualization platform. Finally, a prototype is designed and accomplished in this paper, called VirtualFuzz, as shown in experiment data, DDoS and Escape vulnerabilities can be discovered effectively in our method, 24 test cases are acquired, in which 18 known cases are evaluated and 6 unknown cases are discovered. Moreover, we have gained 3 vulnerability authentications by CVE. In addition, the optimized results for efficiency are emphasized via comparison between VirtualFuzz and other Fuzzing tools.

Key words: Self-adapting; Fuzzing test; Gray Markov Model

通讯作者：肖甫，xiaof@njupt.edu.cn

基金项目：国家自然科学基金项目(No.61373137, 61572260, 61702283)，江苏省高校自然科学研究计划重大项目(14KJA520002); 江苏省杰出青年基金项目（BK20170039）。

Foundation Items: The National Natural Science Foundation of China (No.61373137, 61572260, 61702283), Major Program of Jiangsu Higher Education Institutions (14KJA520002)and Science Foundation for Outstanding Young Scholars of Jiangsu Province (BK20170039).

云计算作为当前的新兴信息技术之一，在学术界及工业界受到广泛关注。其独有的广泛性、通用性及可扩展性为 IT 服务创造了极大的提升空间，以多元化的计算环境满足用户的定制化需求，实现资源的广域共享和高效利用。然而，伴随用户数量及服务内容的高速增长，其所面临的安全问题也日益凸显，面向云计算平台的恶意攻击及信息窃取更是层出不穷，已成为制约其发展的主要因素。以当前典型的云计算环境为例，相关安全威胁可总结如下：

按照云平台提供的服务种类，云计算架构具体可分为：SaaS（Software as a Service，软件即服务）层、PaaS（Platform as a Service，平台即服务）层和 IaaS（Infrastructure as a Service，基础设施即服务）层。SaaS 层面向租户部署应用软件环境，提供定制化的服务以满足租户需求。PaaS 层重在向租户提供可拓展的基础平台，供用户开发、调试自组应用程序。IaaS 层作为云服务的最底层，主要提供支持上传服务的各种虚拟化运行环境及平台，因而该层的安全性尤为重要，其所面临的安全威胁也异常突出[1][2][3]。可造成的威胁场景主要包括：第一，敏感信息泄露。鉴于虚拟环境之间的隔离机制存在漏洞，恶意租户可穿透隔离访问其他租户的应用环境，窃取敏感信息及隐私数据。第二，拒绝服务攻击。攻击者可针对底层虚拟化实现过程的脆弱点发起分布式拒绝服务，导致宿主机宕机。第三，逃逸攻击。攻击者可通过特殊漏洞将自身的代码执行权限从客户机提升至宿主机，充分掌控 Hypervisor 的同时也可对其他虚拟机执行任意命令或操作。目前主流的虚拟化平台包括 Qemu，KVM，Xen 以及 Vmware[4]、Bochs[5]等等。因此，亟待提出一种针对云计算中 IaaS 层的漏洞挖掘与分析方法，针对拒绝服务及逃逸攻击分析虚拟化过程中的漏洞与缺陷。

1. 研究现状

本文实现了一种适用于 IaaS 层中虚拟化平台的漏洞挖掘方法，因此相关的研究工作主要包括以下几方面：第一，软件漏洞分析方法；第二，虚拟化安全分析与加固方法。具体内容分述如下。

1.1 漏洞分析方法

文献[6]定义了软件漏洞和软件漏洞分析技术，并基于两大基础定义提出了软件漏洞分析的技术体系，进而对现有的各种漏洞分析技术进行分类与对比，并展望了该领域的发展方向。具体来看，作者根据分析对象与漏洞形态将分析技术体系分为：软件架构分析、静态分析、动态分析、动静结构分析以及漏洞定位分析等等。类似地，文献[7]综述了程序分析和软件漏洞检测的基本概念、核心问题和传统手段，重点介绍了动态符号执行、白盒模糊测试及相关的实现工具等等。

具体来看，静态分析主要从框架和结构上对目标程序进行分析并理解其中函数、数据入口的相关含义，可有效建立未知软件的理解框架。如文献[8]提出一种协同式逆向推理方法生成目标程序中的静态相关路径，可有效解决关键节点路径引导信息缺失所导致的冗余或无效路径问题。而文献[9]基于语法分析树设计并实现一种软件脆弱性代码克隆检测方法，对目标代码进行基于多种克隆类型的聚类计算，达到了从被测软件代码中检测脆弱代码的目的。另外，符号执行作为静态分析的主流方法，受到工业界及学术界安全研究者的广泛关注[10][11]。但近年的研究学者普遍认为，传统的符号执行方法通常产生大量路径，最终导致路径爆炸，对目标程序分析带来极大困扰。同时由于路径交叉及参数复杂度问题，往往导致最终无法约束求解。但符号执行方法确实在程序路径分析上有较好的覆盖率及自动化效果，因此目前主流的符号执行方法偏向于程序真实执行结合选择性符号执行，用真实执行过程修正静态符号执行过程中出现的错误。文献[12]即利用动态符号执行方法设计并实现面向 Linux 平台的二进制程序危险操作挖掘方法，充分发现目标程序中由于危险操作导致的潜在缺陷或漏洞。文献[13]同样将真实执行与符号执行相结合，提出一种目标制导的混合执行测试方法，可在更短时间内发现目标程序更多缺陷或漏洞。类似地，文献[14]基于混合执行提出一种内存泄漏警报的自动化确认方法，可对静态内存泄漏警报进行有效分类，显著降低人

工确认的工作量。具体到 IaaS 层虚拟化过程中拒绝服务或逃逸类漏洞，单纯的静态分析方法（如路径关联分析）无法准确解析出动态数据执行后的拒绝服务或逃逸效果，而符号执行方法在 IaaS 虚拟化过程的代码量中面临路径爆炸的典型问题。因此，如何将静态分析方法中对程序结构及框架的分析优势整合到 IaaS 层的漏洞挖掘中，是该领域的研究难点之一。

动态分析中较为典型的技术主要包括：程序插桩技术、污点分析技术、模糊测试技术等等。文献[15]基于污点分析与目标程序插桩溢出一种针对信息流的整数漏洞插桩方法，将分析对象约减为污染信息流路径上的所有危险整数操作，大大降低静态插装密度。文献[16]针对 Java 程序中频繁的堆操作设计并实现插桩分析方法，实现虚拟机自适应欲取框架，根据程序运行时的反馈自适应地调整插桩并进行预取优化。而污点分析方法作为程序插桩的后继分析补充，可有效实现动态分析过程中真实数据约束条件的收集，为模拟程序状态及匹配漏洞模式提供极大参考。文献[17][18]综述并设计污点分析方法在漏洞分析场景中的应用，分别在 Android 系统及桌面操作系统中实现隐私泄露检测及访问控制方法。而模糊测试（又称 Fuzzing 测试）是另一种典型的软件安全测试技术，它使用随机的数据作为程序的输入对目标程序进行测试，随后对目标程序的运行状态进行监控，通过观察是否出现崩溃或其他异常的场景，如 CPU 利用率过高，系统假死或简单的程序 bug，以及潜在的安全漏洞等等[19][20][21]。目前的模糊测试主要分两类，一类是基于变异的模糊测试方法，即在已有输入的基础上进行修改来创建新的输入；一类是基于生成的模糊测试方法，即根据产生模型重新生成输入数据，放入目标系统或软件中进行测试。相比程序插桩及污点分析技术，模糊测试可在短时间内获得大量程序异常或崩溃场景，漏洞挖掘效率较高，但需经人工确认排查大量误报过程。文献[22]提出基于快速内存的模糊测试方法，结合静态分析及污点跟踪技术后，充分提高模糊测试精度，可有效发掘二进制程序漏洞。而文献[23]提出基于异常分布导向的智能模糊测试方法，针对目标二进制程序建立样本构造模型，并在迭代测试中不断更新模型参数，抽取已发现程序异常信息对新测试样本的指导意义，在实验中获得若干未知漏洞且性能开销表现较好。同样基于若干动态分析方法的融合，类似工作也在漏洞发现及安全建模方面取得较好效果[24][25]。目前尚无针对 IaaS 中各种虚拟化产品通用性较好的模糊测试方法，单纯基于变异或生产的测试过程无法准确覆盖拒绝服务及逃逸等漏洞特征。

1.2 虚拟化安全分析及加固方法

针对 IaaS 层中虚拟化过程的安全分析或加固方法同样与本文工作有较大相关度。文献[26-28]针对云计算环境下的安全问题进行综述，从云环境的层次结构上梳理相关安全威胁及防御加固方法，重点强调了虚拟化过程的安全威胁，该部分的安全防御被认为是云计算安全防御的基础。而文献[29]针对虚拟化软件栈进行安全研究，分析其中不同软件层的安全威胁、攻击方式和威胁机理，并针对相关安全威胁以可信基为视角，从虚拟机监控器、微虚拟机监控器等方面比较相关安全方案，并指出存在的安全问题。文献[30]详细描述了一种典型的虚拟机逃逸漏洞，解释并实现了从 GuestOS 到 HostOS 的逃逸原理、逃逸步骤、逃逸结果，并完成漏洞申报与工具开发。另外，文献[31]重点关注虚拟机自省功能中潜在的安全风险及安全防御方法，介绍如何基于虚拟机自省技术实现 GuestOS 对 HostOS 的操作感知，并深入分析攻击者面向该技术的绕过方法，最终也提出了一些安全加固思想及方案。由此可见，IaaS 层中相关安全研究主要集中在虚拟化过程中各种典型的应用机制及应用场景中，如虚拟化软件栈、虚拟机自省等，而针对虚拟化逃逸或拒绝服务漏洞尚无通用化、概括性的理论分析方法。

基于以上分析可知，单纯的静态分析方法可能引发如路径爆炸或路径不可达等典型问题，因而目前主流的静态分析需引入部分动态数据执行过程进行修正，如混合执行方法。而污点跟踪及模糊测试作为动态分析的主流方法，在多种系统环境及目标设备上获得较好的漏洞挖掘效果。具体到虚拟化逃逸及拒绝服务类漏洞，面前尚无形式化且有效的漏洞挖掘及分析方

法,单纯的静态分析显然不足以覆盖虚拟化逃逸及拒绝服务的动态数据执行特征，鉴于虚拟化实现过程的复杂度，需采用一定方法缩小漏洞挖掘的范围，因此普通的模糊测试方法亦不适用。因此，为解决针对虚拟化逃逸及拒绝服务漏洞的自动化挖掘问题，本文面向 IaaS 层的几种典型虚拟化平台展开此类漏洞的挖掘与分析，针对威胁较大的拒绝服务类与逃逸类漏洞描述漏洞判决规则，使用自适应模糊测试方法定位判决规则在源码中的路径约束条件，最终回溯收集所得的崩溃场景生成漏洞利用程序，为验证该方法的有效性，我们设计并实现了基于此方法的原型系统，通过若干已知漏洞的验证及未知漏洞的挖掘证明在真实逃逸及拒绝服务漏洞场景下自适应模糊测试的有效性。

2. 漏洞原理

为引出后继工作的相关内容，本节针对 IaaS 层中虚拟化漏洞的相关背景及原理进行简单介绍。以最典型的毒液漏洞为例（CVE-2015-3456），该漏洞存在于虚拟软盘驱动器的实现代码中，可允许攻击者从受感染虚拟机（GuestOS）中摆脱访客身份限制，可轻易获取主机（HostOS）的代码执行权限。同时攻击者还可以利用它访问主机系统以及主机上运行的所有虚拟机，并能够提升重要的访问权限。如图 1 所示，条件 1：(fdctrl->fifo[fdctrl->data_pos - 1] & 0x80)可以被构造成功，而 fdctrl->data_len 最大只能为 6，因此该条件错误，则系统运行流程进入第 1 个 if 分支，且绕过第 2 个 if 分支，且从第 1 个 if 分支中走出后同样可绕过 else if 分支，则 fdctl_set_fifo()被绕过，目标缓冲区不会被重置，可被溢出写入。刚好在堆空间的布局上存在一个 HostOS 将调用的指针，因此精确控制缓冲区中的值，可成功覆盖该指针，得到代码执行权限从 GuestOS 到 HostOS 的目的。

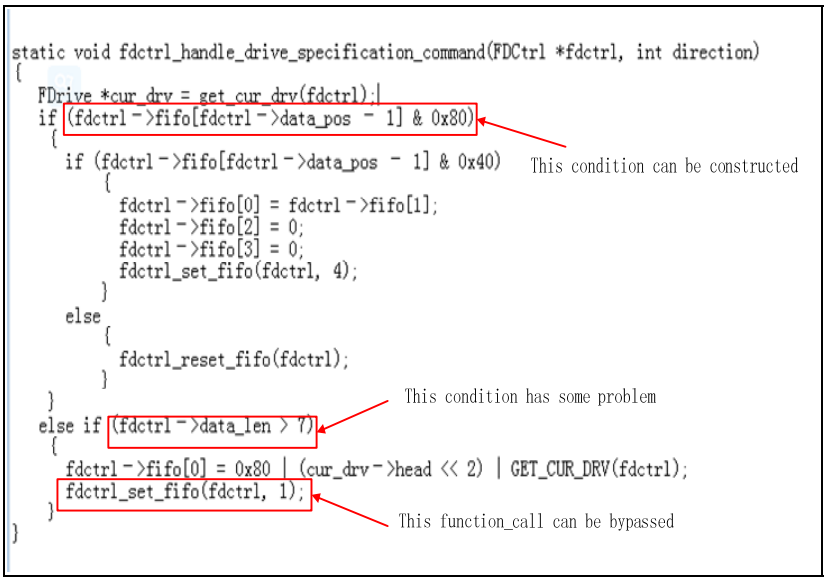

图 1 逃逸漏洞细节：CVE-2015-3456

3. 相关定义

定义 1. 目标缓冲区：Target_Buffer，简记为 TB，TB={CPU, Memroy, IO}。该定义用于描述目标漏洞相关联的目标缓冲区，包括用于存放在堆栈上的各种缓冲区定义。作为漏洞挖掘过程中的主要研究对象，该定义赋值基于静态源码审计及动态数据注入获得，根据虚拟化实现过程中的具体特征共分 3 类，分别是 CPU 类目标缓冲区、Memory 类目标缓冲区及 IO 类目标缓冲区。

定义 2. 目标函数：Target_Func，简记为 TF，TF={CPU, Memroy, IO}。该定义用于描述目标漏洞相关联的目标函数，根据虚拟化实现过程中的具体特征共分 3 类，分别是 CPU 类目标函数、Memory 类目标函数及 IO 类目标函数。

定义 3. 目标数据块：Target_Datablock，简记为 TD，TD={CPU, Memroy, IO}。该定义用于描述目标漏洞相关联的目标数据块，根据虚拟化实现过程中的具体特征共分 3 类，分别是 CPU 类目标数据块、Memory 类目标数据块及 IO 类目标数据块。

定义 4. 目标指针：Target_Pointer，简记为 TP，TP={CPU, Memroy, IO}。该定义用于描述目标漏洞相关联的目标指针，根据虚拟化实现过程中的具体特征共分 3 类，分别是 CPU 类目标指针、Memory 类目标指针及 IO 类目标指针。

定义 5. 数据约束入口：Constrained_Emport，简记为 CEm，CEm={Crossed_Func, Crossed_Module, Crossed_File}。该定义用于描述崩溃及劫持点相关的数据约束条件入口，主要指控制流到达目标漏洞崩溃点及劫持点过程的入口位置。其具体赋值过程为崩溃场景中的数据约束入口地址，根据入口地址与崩溃地址的比对主要分成：第一，跨函数类的数据约束入口；第二，跨模块类的数据约束入口；第三，跨文件类的数据约束入口。

定义 6. 数据约束路径：Constrained_Path，简记为 CP，CP={short, medium, long}。该定义用于描述崩溃及劫持点相关的数据约束条件路径，主要指控制流从数据约束入口开始的数据构成条件。以构造的平均路径长度为对比指标，可分为 3 类：第一，短约束路径；第二，中约束路径；第三，长约束路径。

定义 7. 数据约束出口：Constrained_Export，简记为 CEx，CEx={Crossed_Func, Crossed_Module, Crossed_File}。该定义用于描述崩溃及劫持点相关的数据约束条件终点，主要指控制流崩溃或劫持前的数据构成结束位置，在数据约束出口点处即开始漏洞触发过程。同样根据出口地址与崩溃地址的比对主要分成：第一，跨函数类的数据约束出口；第二，跨模块类的数据约束出口；第三，跨文件类的数据约束出口。

定义 8.疑似 DDoS 点：Suspected_DDoS_Point，简记为 SDP，SDP={Pointer_DDoS, Data_DDoS, InfiniteLoop_DDoS}。该定义用于描述可判决为拒绝服务类型的疑似崩溃点，通常的漏洞类型中若可导致目标系统控制流崩溃，及可定义为拒绝服务漏洞；但在 IaaS 的虚拟化环境中，只有宿主机的控制流被虚拟机中输入数据攻击造成崩溃才可被认定为 DDoS 漏洞。该定义主要用于完成该过程的判决，其赋值主要根据疑似崩溃点到疑似 DDoS 点的转换过程，具体分为：第一，指针导致的 DDoS；第二，数据导致的 DDoS；第三，无限循环导致的 DDoS。

定义 9. 疑似逃逸点：Suspected_Escape_Point，简记为 SEP，SEP={Pointer_Escape, Func_Escape, Data_Escape}。该定义用于描述可判决为逃逸类型的疑似劫持点，通常的漏洞类型中若可导致目标系统控制流劫持，及可定义为高危漏洞；但在 IaaS 的虚拟化环境中，只有宿主机的控制流在虚拟机中被劫持，才可被认定为逃逸漏洞。该定义主要用于完成该过程的判决。根据从疑似劫持点到疑似逃逸点的转换过程，赋值主要分成：第一，利用指针进行逃逸；第二，利用函数进行逃逸；第三，利用数据进行逃逸。

定义 10.可 DDoS 点：DDoSed_Point，简记为 DP，DP={Stable_Pointer_DDoS, Stable_Data_DDoS, Stable_InfiniteLoop_DDoS}。该定义用于描述确定可导致虚拟机到宿主机的拒绝服务攻击点。该定义用于描述确定可导致虚拟机到宿主机的拒绝服务攻击点。Fuzzing 测试过程中收集所得的疑似 DDoS 点同样有一定的偶然性，结合静态源码审计确定疑似 DDoS 点在人工构造数据约束条件下的稳定漏洞触发，因此定义为可 DDoS 点。根据从疑似 DDoS 点到可 DDoS 点的转移过程对其进行复制，主要包括：第一，稳定的可用指针；第二，稳定的可用数据；第三，稳定的可用无限循环。

定义 11. 可逃逸点：Escaped_Point，简记为 EP，EP={Stable_Pointer_Escape, Stable_Func_Escape, Stable_Data_Escape}。该定义用于描述确定可导致虚拟机到宿主机的逃逸攻击点。Fuzzing 测试过程中收集所得的疑似逃逸点有一定的偶然因素，需结合静态源码审计确定疑似逃逸点在人工构造数据约束条件下的可用性及稳定性，因此定义为可逃逸点。

根据从疑似逃逸点到可逃逸点的转移过程对其进行复制，主要包括：第一，稳定的可用指针；第二，稳定的可用函数；第三，稳定的可用数据。

定义 12. 虚拟化拒绝服务漏洞：IaaS_DDoS_Vul，该定义用于描述面向虚拟化过程的拒绝服务漏洞，包括漏洞利用的目标对象、数据约束入口、数据约束路径、数据约束出口、可 DDoS 攻击点等漏洞相关特征，具体可表示为：

IaaS_DDoS_Vul={TB, TP, TD, CEm, CP, CEx, DP}.

定义 13. 虚拟化逃逸漏洞：IaaS_Escape_Vul，该定义用于描述面向虚拟化过程的逃逸漏洞，包括漏洞利用的目标对象、数据约束入口、数据约束路径、数据约束出口、可逃逸攻击点等漏洞相关特征。具体可表示为：

IaaS_Escape_Vul={TB, TP, TF, TD, CEm, CP, CEx, EP}.

根据以上基本定义，系统面向 IaaS 的虚拟化平台部署污点分析和内存切片，经数据采集、数据清洗、数据抽象、数据聚类后可对以上定义赋值。而漏洞模式属于高层语义，如需将基本定义中的若干属性或特征进行聚合，为此需总结和设计从低层语义（如基本定义）到高层语义的判决规则。该规则从低层语义的取值中对操作语义建模，进而基于操作语义为高层语义建模，给出细粒度、全局化的漏洞模式赋值。该映射规则的正确性经若干种主流虚拟化平台下动态污点跟踪及源码静态分析所验证，从而确保推求所得的上层语义的正确性与合理性。

规则 1. 针对目标缓冲区，数据约束入口与崩溃地址距离为跨函数或跨模块，数据约束路径为短或中距离路径，数据约束出口为跨函数或跨模块，有稳定可用的数据或无限循环时，存在面向目标缓冲区的虚拟化拒绝服务漏洞：

if TB=CPU|Memory|IO && CEm=Crossed_Func|Crossed_Module && CP=short|medium && CEx=Crossed_Func|Crossed_Module && DP=Stable_Data_DDoS|Stable_InfiniteLoop_DDoS,

then IaaS_DDoS_Vul$_{TB}$

规则 2. 针对目标指针，数据约束入口与崩溃地址距离为跨函数或跨模块，数据约束路径为短或中或长距离路径，数据约束出口为跨函数或跨模块，存在稳定可用的指针或数据覆盖指针时，存在面向目标指针的虚拟化拒绝服务漏洞：

if TP=CPU|Memory|IO && CEm=Crossed_Func|Crossed_Module && CP=short|medium|long && CEx=Crossed_Func|Crossed_Module && DP=Stable_Pointer_DDoS|Stable_Data_DDoS,

then IaaS_DDoS_Vul$_{TP}$

规则 3. 针对目标缓冲区，数据约束入口与崩溃地址距离为跨函数或跨模块，数据约束路径为短或中距离路径，数据约束出口为跨函数或跨模块，有稳定可用的数据覆盖指针或函数时，存在面向目标缓冲区的虚拟化逃逸漏洞：

if TB=CPU|Memory|IO && CEm=Crossed_Func|Crossed_Module && CP=short|medium && CEx=Crossed_Func|Crossed_Module && DP=Stable_Data_Escape,

then IaaS_Escape_Vul$_{TB}$

规则 4. 针对目标函数，数据约束入口与崩溃地址距离为跨模块或跨文件，数据约束路径为中或长距离路径，数据约束出口为跨模块或跨文件，有稳定可用的函数时，存在面向目标函数的虚拟化逃逸漏洞：

if TF=CPU|Memory|IO && CEm=Crossed_Module|Crossed_File && CP=medium|long && CEx=Crossed_Module|Crossed_File && DP=Stable_Func_Escape,

then IaaS_Escape_Vul$_{TF}$

规则 5. 针对目标指针，数据约束入口与崩溃地址距离为跨函数或跨模块，数据约束路径为短或中距离路径，数据约束出口为跨函数或跨模块，有稳定可用的指针时，存在面向目标指针的虚拟化逃逸漏洞：

if TP=CPU|Memory|IO && CEm=Crossed_Func|Crossed_Module && CP=short|medium

&& CEx=Crossed_Func|Crossed_Module && DP=Stable_Pointer_Escape,

then IaaS_Escape_Vul$_{TP}$

规则 6. 针对目标数据区，数据约束入口与崩溃地址距离为跨函数或跨模块，数据约束路径为短或中距离路径，数据约束出口为跨函数或跨模块，有稳定可用的数据区时，存在面向目标数据区的虚拟化逃逸漏洞：

if TD=CPU|Memory|IO && CEm=Crossed_Func|Crossed_Module && CP=short|medium && CEx=Crossed_Func|Crossed_Module && DP=Stable_Data_Escape,

then IaaS_Escape_Vul$_{TD}$

4. 基于灰度马尔科夫链的模糊测试方法

给定一个随机过程 X_t，如已知随机过程在某一时间点 t_0 所处的状态，过程 X_t 在以后的时间段内状态与过程 t_0 时刻之前的状态情况没有关系。这种在已知当前条件下，过程将来的变化跟过去没有关系的性质，称为马尔科夫性，而具有这种无后向效果的随机过程叫做马尔科夫过程[32][33]。而利用马尔科夫过程构建的马尔科夫链通常用于预测此类过程的发展趋势。通常所用的预测方法包括：趋势预测法、回归预测法、灰系统预测法以及马尔科夫预测方法等等。通常这几种方法各有优缺点，趋势预测法计算简单，但准确性较低，预测结果偏差较大；灰色预测主要用于短期预测场景，缺点在于对长期预测效果较差，且针对随机性较大的数据集拟合效果较差；马尔科夫链预测面向长期预测及随机波动性较大的数据列有很好的预测效果，但马尔科夫链预测对象要求具有平稳过程。本文在面向 IaaS 层部署模糊测试的过程中引入随机预测，其主要目的在于指导模糊测试的方向，将预测结果引入到实时的模糊测试监督过程中，通过与漏洞特征的判决规则进行比对，可有效纠正错误或无效的模糊测试过程，避免模糊测试过程中所产生的死循环状态及计算资源消耗状态，达到自适应模糊测试的效果。因此，为提高预测精度，需充分模拟 IaaS 层中的虚拟化运行状态，考虑将灰系统预测方法与马尔科夫链预测方法相结合，形成灰度马尔科夫链预测模型。该方法可借助灰系统模型预测的变化趋势，同时可基于马尔科夫链预测状态规律，从而达到无缝模拟虚拟化平台运行状态的优良效果。具体过程如下。

4.1 建立灰模型系统

灰模型系统通常基于灰色 GM(1,1)模型建立，主要计算 $\hat{Y}(k)$ 曲线，具体计算过程如下所示：

针对非负原始序列：

$$X^{(0)}=\left\{x^{(0)}(1),x^{(0)}(2),...,x^{(0)}(n)\right\}$$

本文中具体的序列需代入：

$$X^{(0)}=\left\{x^{(0)}.TB,\ x^{(0)}.TP,\ x^{(0)}.TF,\ x^{(0)}.TD,\ x^{(0)}.CEm,\ x^{(0)}.CP,\ x^{(0)}.CEx,\ x^{(0)}.EP\right\}$$

对 $X^{(0)}$ 作一次累加：

$$x^{(1)}(k)=\sum_{i=1}^{k}x^{(0)}(i)\ ;\ \text{k=1,2,…,n}$$

得到生成数列为：

$$X^{(1)}=\left\{x^{(1)}(1),x^{(1)}(2),...,x^{(1)}(n)\right\}$$

省略后继的白化微分方程及离散化等过程，计算 $\hat{Y}(k)$ 曲线为：

$$\hat{x}^{(0)}(k+1)=\hat{x}^{(1)}(k+1)-\hat{x}^{(1)}(k)=(1-e^{\hat{a}})\left[x^{(1)}(1)-\frac{\hat{u}}{\hat{a}}\right]e^{-\hat{a}k}$$

4.2 划分目标状态

以 $\hat{Y}(k)$ 为标准，将预测对象的序列划分为平行于 $\hat{Y}(k)$ 的若干区域，每个区域用 Q_i 表示，其中又包含若干个子状态：

$Q_i=[Q_i^{\alpha},Q_i^{\beta}]$,

$Q_i^{\alpha}=\hat{Y}(k)+\alpha_i$,

$Q_i^{\beta}=\hat{Y}(k)+\beta_i$.　i=1,2,…,n

其中 α_i 和 β_i 根据虚拟化平台运行状态中疑似 DDoS 点 SDP 及疑似逃逸点 SEP 生成，用于计算逼近 DDoS 点及逃逸点的状态取值。

4.3 转移概率矩阵

状态转移可理解为虚拟化平台运行过程中记录的离散化状态取值的转移过程，即目标集合 $X=\{x.TB, x.TP, x.TF, x.TD, x.CEm, x.CP, x.CEx, x.EP\}$，而转移规则需要被动地适应目标虚拟化平台中采样值的变化趋向。转移概率矩阵表示为：

$P_{ij}^{(m)}=M_{ij}^{(m)}/M_i$, i,j=1,2,…,n

P_{ij} 表示目标系统从 Q_i 经 m 步骤转移到 Q_j 的概率，M_i 表示原始状态集合按照一定的概率落入状态 Q_i 的样本数量。而 $M_{ij}^{(m)}$ 则表示状态 Q_i 经过 m 步骤转移到 Q_j 的原始数据集合样本数量。

4.4 生成预测值

若未知的转移状态表示为 Q_u，则预测值的变化区间为 $Q_u=[Q_u^{\alpha},Q_u^{\beta}]$，预测值 $\hat{Y}(k)$ 取该区间的中点，则 $\hat{Y}(k)=\frac{Q_u^{\alpha}+Q_u^{\beta}}{2}$。

4.5 检验预测精度

预测精度主要通过后验差比值及小误差频率来完成，后验差比值生成 C 值越小越好，而小误差频率 P 值越大越好，需考虑 C<0.35 且 P>0.95。

4.6 拟合预测方向

以上预测过程的主要目的是通过该模型获取目标虚拟化平台中模糊测试数据的生成方向及内容，考虑以下几种情况作为预测方向的拟合特征：

1. $\hat{Y}(k)_i \rightarrow IaaS\_DDoS\_Vul_i$，$i\in\{TP,TD,TF,TB\}$

   //存在预测方向拟合虚拟化拒绝服务规则

2. $\hat{Y}(k)_i \rightarrow IaaS\_Escape\_Vul_i$，$i\in\{TP,TD,TF,TB\}$

   //存在预测方向拟合虚拟化逃逸规则

3. $\hat{Y}(k)\rightarrow IaaS\_Escape\_Vul$ and $\hat{Y}(k)\rightarrow IaaS\_DDoS\_Vul$，$i\in\{TP,TD,TF,TB\}$

   //存在预测方向同时拟合虚拟化拒绝服务规则与逃逸规则

4.7 指导模糊测试

本文模糊测试所采用的基本方法是面向虚拟化平台中 3 种典型的虚拟化过程展开，包括 CPU 虚拟化、内存虚拟化及 IO 虚拟化，首先基于源码静态审计的方法确定各虚拟化功能模块的函数调用关系、函数调用接口、函数参数约束条件等等，而后通过动态调试分析虚拟化过程中的动态函数调用路径及函数执行结果，并通过与静态源码的比对确定函数相关信息。而后依次遍历确定后的目标测试函数，并将已收集的函数入口参数取值范围及多重函数调用过程作为模糊测试对象，基于标准化的随机化种子生成方法在参数取值范围内及多重函数调用路径间生成模糊测试用例，而后大量冲击目标系统，观察系统反馈结果。而以上基于灰度马尔科夫链的预测模型将生成系统状态预测值，通过预测方向的拟合结果，可判决当前虚拟化模块下的模糊测试方向，从而决定是否在拟合的漏洞发现预测方向下继续执行模糊测试，或自适应调整当前模糊测试输入参数及函数调用关系，使得计算后的预测方向与正确的拟合方向相同。针对不同虚拟化模块中的不同虚拟化功能，在决定自适应调整后需改变不同的参数及不同的函数调用关系，因此该过程需面向虚拟化实现机制加入部分人工干预。

5. 原型系统实现

本文面向 IaaS 层的几种典型虚拟化平台部署并实现原型系统 VirtualFuzz，主要基于 GuestOS 与 HostOS 的通信模型进行源码审计、动态调试与污点分析，经过源代码的静态审计获取静态函数调用关系与静态关联数据结构，在动态调试过程中补充过程间调用及动态填充数据结构，基于污点分析修正动态函数调用关系及输入数据约束条件在未知输入数据约束条件下的异常情况，最终按模块、直接调用函数、间接调用函数、过程间调用函数的顺序依次完成模糊测试过程。其中用到的源码审计数据来源主要取自各实验版本中开源项目（vmware、bochs 为非开源项目，因此该两类实验对象中静态分析主要通过逆向分析完成），动态调试主要使用 windows 平台及 linux 平台下典型的调试工具：windbg（版本号：6.12.2.633）、GDB（版本号：7.3）等，污点跟踪则主要通过静态分析工具插件开发（IDA6.8 版本）、调试器插件编写（windbg 插件）、自研的内存查看及标记工具完成。核心功能中的模糊测试部分则是根据以上自适应指导过程具体在各 IaaS 层虚拟化产品中的网络数据包接口、外部命令接口、CPU 指令接口中展开模糊测试，挖掘虚拟化过程在接受外部网络数据包、HostOS 与 GuestOS 通信命令、CPU 指令集实现过程中的拒绝服务与逃逸漏洞。同时，基于灰度马尔科夫为模糊测试过程生成预测值，实时修正模糊测试的输入及方向，避免错误或低效的测试开销。具体过程如图 2 所示。

由图 2 可知，系统首先对实验对象的开源代码或二进制文件展开词法分析及语法分析，并通过抽象语法树的方法获得树形表达结果，为转化为中间语言表达提供支持。而后基于 Valgrind 中的 VEX-IR 中间语言表达实现目标数据集合的污点标记，并与 Windows 及 Linux 平台下的动态调试器插件进行通信，完成动态污点跟踪过程。随后针对 3 种特征调用展开模糊测试，对崩溃场景中寄存器取值进行收集，最终生成漏洞利用程序（PoC，Proof of Concept）。

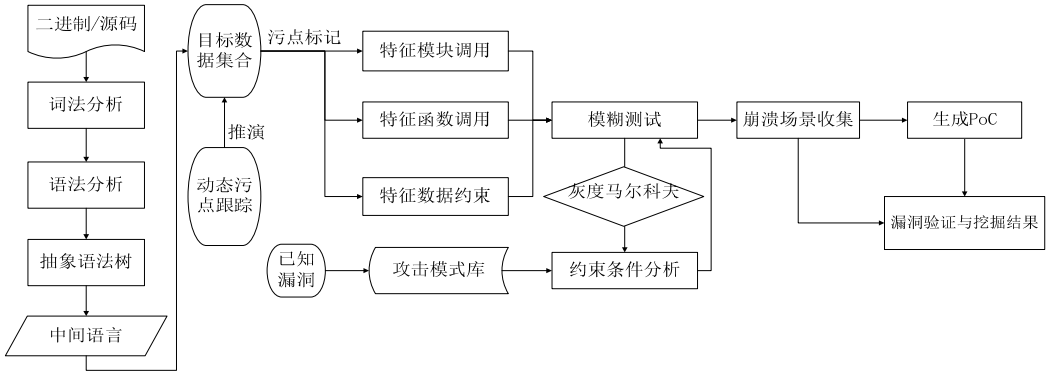

图 2 VirtualFuzz 设计流程图

6. 实验

实验主要针对 5 类应用范围最广泛的 IaaS 层中虚拟化平台及工具展开测试，包括 Xen、KVM、Qemu、Bochs、Vmware，将跨平台覆盖虚拟化实现过程中大量关键技术。首先基于静态与动态分析过程建立目标测试数据集，并在此基础上拓展建立面向拒绝服务与逃逸的攻击模式库；而后部署所设计的模糊测试系统，基于攻击模式库依次遍历目标测试数据集中的关键控制流与数据结构，并实时生成灰度马尔科夫预测值，比对判决规则后指导模糊测试方向，达到自适应效果。具体的实验环境参数如下所示：

1）硬件平台：

处理器：Intel(R)Core2Duo CPU E7500@2.13GHz

内存：32.00GB

显卡：NVIDIA Ge Force 9600 GSO 512

2）软件平台：

操作系统：Windows 7 旗舰版 Service Pack1；SUSE Linux Enterprise Server 12

开发环境：VS 2013；GCC 7.1

3）测试对象：

Xen: 3.0.1 -- 4.5.x（基于 Linux 平台）

KVM: 11 – 17（基于 Linux 平台）

Qemu: 2.2.1 -- 2.7.x（基于 Linux 平台）

Vmware: 10.0.4 -- 12.0.x（基于 Windows 平台）

Bochs: 2.5.0 -- 2.6.x（基于 Windows 平台）

6.1 目标测试数据集

实验基于目标平台的静态与动态分析进行类型推演，收集关键函数、数据结构及相互间的交叉关系，尤其针对已曝光漏洞的功能模块及相关函数，需基于漏洞分析生成原始的目标测试结合，类型推演后去除已曝光的数据约束关系，将相关信息添加到目标测试数据集中。此处以漏洞 CVE-2015-3456 为例，展示如何从静态与动态分析及漏洞场景分析中生成目标测试集合，并完成类型推演，具体过程如图 3 所示。

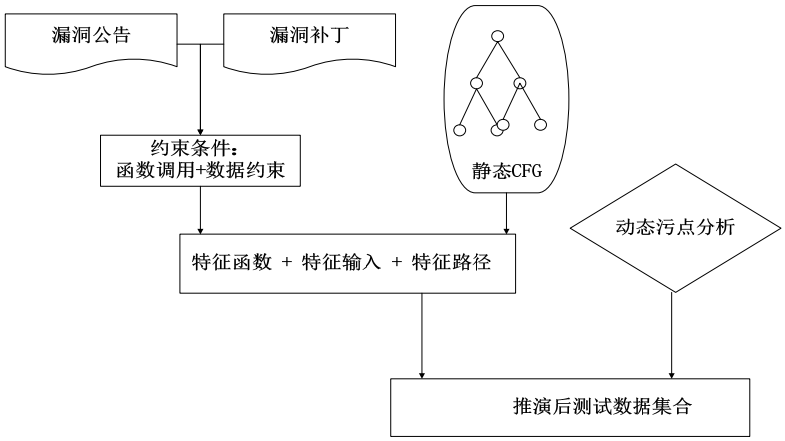

图 3 目标测试集合生成过程

6.2 自适应模糊测试过程分析

工具针对 9.1 节中推演所得的目标测试集合，展开自适应的模糊测试过程。面向 9.1 节中推演所得的目标测试集合，逐个平台、逐个模块、逐个函数及逐个约束入口进行模糊测试，同时计算基于灰度马尔科夫的预测值，已拟合拒绝服务及逃逸漏洞判决规则为指导，调整模糊测试方向。鉴于目标测试数据集中元素数量庞大，本文对各平台下的不同虚拟化模块进行

测试方向与预测过程的统计分析，具体统计数据如图 4 所示。

由图 4 可知，实验对 4 类平台中的自适应模糊测试过程进行数据统计，将目标系统中共 486538 个采样点进行分类，并选择其中有限的 100 个采样点进行趋势绘图，其取值来自于 $X=\{x.TB, x.TP, x.TF, x.TD, x.CEm, x.CP, x.CEx, x.EP\}$ 集合在多属性决策方法中的理想优基点法下的取值，取值从 0 到 1，根据时间点 N 下几个具备典型漏洞特征的采样点作为记录，可绘制如图 4 所示的模糊测试方向取值点及灰度马尔科夫预测取值点。具体来说，以 Xen 虚拟化平台为例，漏洞特征规则下的模糊测试采样点大都分布在 0.25 到 0.6 之间，而马尔科夫预测点与其的误差在[0.13,0.22]内，针对第 3 采样点处出现最大误差，该点在预测模型下显示为无漏洞特征，而模糊测试过程中判决为疑似拒绝服务漏洞。后经人工排查，该点处的拒绝服务漏洞不具备可用的数据约束路径。同理可分析 Qemu 平台下的对比结果，在所展示的第 7 采样点处，多属性聚合值为最小，取值为 0.572，表征在理想优基点模型中，该点的漏洞特征与其他疑似漏洞相比最小，即最不可能判决为漏洞。但该点处模糊测方向与预测方向拟合效果较好，误差仅为 0.08，且在预测模型中被判决为疑似拒绝服务漏洞。后经人工排

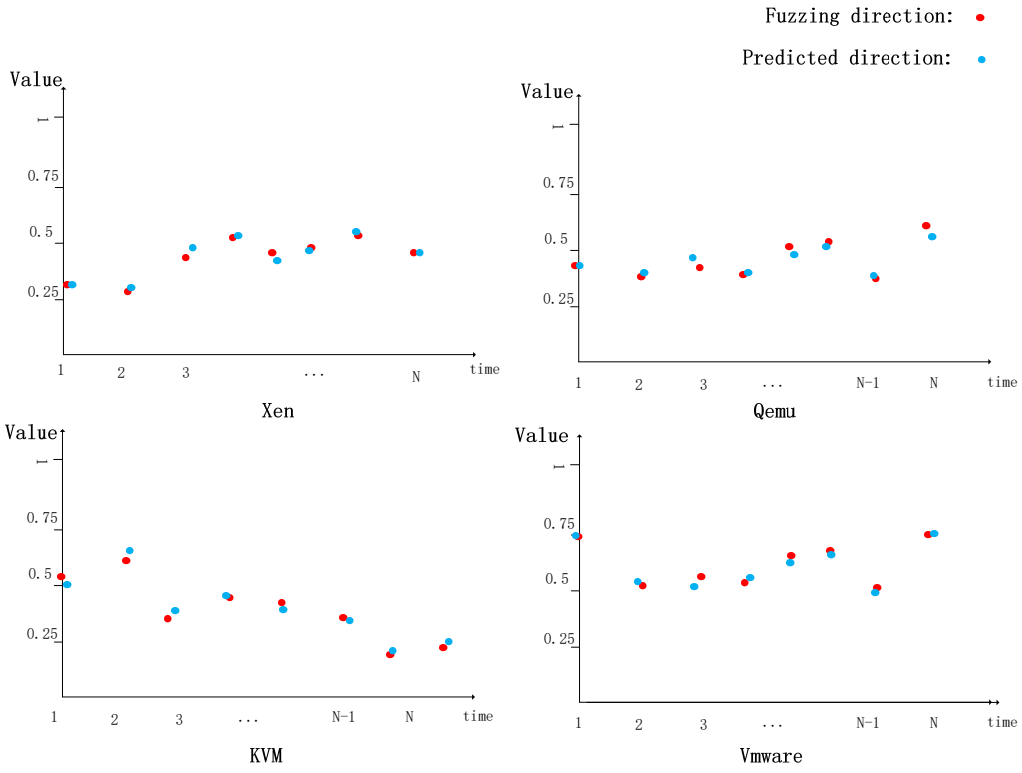

图 4 自适应模糊测试预测统计

查可知该点处无法触发拒绝服务漏洞利用效果，证明实际的模糊测试所生成的数据约束路径无效。同理对比分析 KVM 和 Vmware 平台，KVM 整体安全性分析结果较差，多处存在差拟合效果的漏洞评估。而 Vmware 平台的拟合效果较好，误差控制在 0.03 到 0.14 之间，可被认为具备较高的安全级别，且模糊测试中判决得到的漏洞个数较少，经人工确认后的可用漏洞更少。同理可展示 Bochs 虚拟化环境下的数据分析结果，其安全性同 Vmware 基本同级，限于篇幅，不再展开讲解。

6.3 漏洞模式判决

对目标系统中通过自适应模糊测试获得的漏洞进行统计并整理，鉴于 Vmware 及 Bochs 平台下并未发现有效漏洞，因此主要统计前 3 类平台（Xen，KVM，Qemu）中的漏洞数量，

主要包括两类：第一，系统认定的漏洞；第二，经人工排查后确定可利用的漏洞。第一类漏洞基于模糊测试中的随机化输入数据及系统崩溃场景记录判决，第二类需在第一类的基础上回溯分析触发场景的数据约束入口，保证在固定的数据约束条件下可生成拒绝服务或逃逸场景。具体数据如表 1 所示，对 3 类平台进行总结，分别获得拒绝服务漏洞场景 5、4、7 个，分别获得逃逸场景 2、2、4 个，分别覆盖 Xen、KVM、Qemu 平台。

表 1 漏洞判决结果

平台 / 种类	Xen	KVM	Qemu
拒绝服务	31(5)	22(4)	54(7)
逃逸	20(2)	25(2)	37(4)

6.4 已知漏洞验证及未知漏洞发现

针对已获得的漏洞个数及类型，本文总结已发现的漏洞如表 2 所示，共计 24 个，包括 Xen 平台中的 7 个，KVM 平台中的 6 个，以及 Qemu 平台中的 11 个，依次命名为 Vul_A 到 Vul_X。由于原型系统 VirtualFuzz 中使用的静态分析结果和攻击模式库均是从开放源码和已知漏洞中推演而来，因此该 24 个漏洞中包含用于建模的若干个已知 CVE 漏洞，同时也发现了大量未知的高风险漏洞。如表所示，系统所挖掘漏洞分别对应到 3 个主流虚拟化平台，已针对 Xen 发现 2 个未知漏洞，分别对应到 IO 模块的网卡处理函数及声卡处理函数中，目前已申请 CVE 号，待审批状态中；KVM 虚拟化环境中主要完成了已知漏洞的验证过程，包括 4 个拒绝服务漏洞及 2 个逃逸漏洞；Qemu 虚拟化中获得了 4 个未知漏洞，其中有 3 个已拿到 CVE 号，另一个是本文工作在最近所发现，因此也在申请 CVE 中，具体模块位于 IO 处理过程的网卡数据收包函数中，其余主要验证了系统中部分的已知漏洞。另外从描述中可知，还有部分漏洞属于已发布但未分配 CVE 号的情况，此类漏洞可通过漏洞描述基本确定所得漏洞是否和其特征相同。

表 2 验证已知漏洞及发现未知漏洞

漏洞名	平台	类型	模块	已知/未知
Vul_A	Xen	拒绝服务	CPU	已知：CVE-2015-7812
Vul_B	Xen	拒绝服务	IO	已知：CVE-2015-0268
Vul_C	Xen	拒绝服务	IO	已知：CVE-2014-2580
Vul_D	Xen	拒绝服务	IO	已知：CVE-2014-2580
Vul_E	Xen	拒绝服务	IO	**未知，已申请 CVE**
Vul_F	Xen	逃逸	Memory	已知：CVE-2015-7835
Vul_G	Xen	逃逸	IO	**未知，已申请 CVE**
Vul_H	KVM	拒绝服务	IO	已知：CVE-2015-6815
Vul_I	KVM	拒绝服务	IO	已知：CVE-2014-7842
Vul_J	KVM	拒绝服务	Memory	已知：CVE-2016-4440
Vul_K	KVM	拒绝服务	IO	已知：无 CVE 号
Vul_L	KVM	逃逸	IO	已知：CVE-2015-6815
Vul_M	KVM	逃逸	IO	已知：CVE-2015-0239
Vul_N	Qemu	拒绝服务	CPU	已知：CVE-2016-2858
Vul_O	Qemu	拒绝服务	Memory	已知：无 CVE 号
Vul_P	Qemu	拒绝服务	IO	**未知：获得 CVE-2016-2841**
Vul_Q	Qemu	拒绝服务	IO	**未知：获得 CVE-2016-0749**

Vul_R	Qemu	拒绝服务	IO	**未知：获得 CVE-2016-2150**
Vul_S	Qemu	拒绝服务	IO	已知：CVE-2014-3672
Vul_T	Qemu	拒绝服务	IO	已知：CVE-2016-2391
Vul_U	Qemu	逃逸	IO	已知：CVE-2016-2391
Vul_V	Qemu	逃逸	IO	已知：CVE-2015-7504
Vul_W	Qemu	逃逸	IO	**未知，已申请 CVE**
Vul_X	Qemu	逃逸	IO	已知：CVE-2015-3456

以 CVE-2016-2841 为例，展开漏洞挖掘的完整过程如下：1）经测试目标推演后，Qemu 源码中测试数据集中网络接口集合主要来自于 hw/net 目录下的各类源码对网卡的模拟，将其输入原型系统，通过词法分析、语法分析、抽象语法树处理后转化为 VEX-IR 中间语言进行表达，基于 GDB 动态调试过程在中间语言表达中插入动态监控点，重点标记特征模块、特征函数及特征数据约束条件。而后基于已知 ne2000 网卡中漏洞的特征点完成相关定义及规则的赋值与判决。在建立灰系统后使用马尔科夫链拟合网卡在接受数据包过程中可能诱发拒绝服务及逃逸的数据约束条件。在面向特征模块、函数及数据条件的随机化测试过程中加入马尔科夫拟合过程的指导作用，使得随机化过程按照灰色马尔科夫拟合方向进行数据变异及数据生成。最终通过构造特征化的 PSTARTPSTOP 寄存器使目标 VM 拒绝服务。

6.5 性能分析与比较

目前工业界及学术界均有较多的模糊测试工具，为衡量 VirtualFuzz 原型系统的性能开销，本文对比了 3 种典型的商业化模糊测试工具，主要计算目标系统在模糊测试运行前后的性能损耗。如表 3 所示，AFL[34]主要面向应用程序展开智能化模糊测试，可有效获得测试中错误位置并绕行，性能损耗约为 46%。而 Trinity[35]是面向 Android 环境的模糊测试工具，可同时兼容 Android 应用层及内核层，性能损耗约为 36%。而 Peach[36]是近年来安全研究人员及漏洞挖掘人员常用的模糊测试工具，适用范围较广，功能强大且界面友好，其性能损耗约为 41%。而本文中的原型系统 VirtualFuzz 面向 5 种虚拟化平台时的平均损耗达到 23.69%，大大优化了已有工具的性能开销，可知基于灰度的马尔科夫自适应过程为系统性能提升做出了巨大贡献，后继还有一定的优化空间。

表 3 原型系统与其他模糊测试工具的性能比较

污点跟踪方案	测试前(us)	测试中(us)	平均性能损耗(%)
AFL	47.2291	88.2306	46.12%
Trinity	66.4417	93.2191	36.61%
Peach	53.6931	79.3378	41.35%
VirtualFuzz	59.8449	81.3215	23.69%

7. 结论

本文面向 IaaS 层中的虚拟化平台设计并实现了一种漏洞挖掘方法，基于静态源码审计及动态污点跟踪实现定向模块及函数的模糊测试，并设计了一种面向模糊测试的自适应监督方法：灰度马尔科夫预测模型，在预测值的引导下实时调整模糊测试的方向和内容，实现有效且快速的虚拟化漏洞验证与挖掘。后继需要深入拓展各模块及函数的调用关系，并进一步优化模糊测试的效率，希望能在不依托已知漏洞的情况下实现更大范围的自适应模糊测试方法。

参考文献

1. ISO. Information technology -- Open Virtualization Format (OVF) specification [EB/OL]. https://www.iso.org/standard/59388.html.

2. ISO.Information technology -- Virtualization Management Specification. [EB/OL].https://www.iso.org/standard/63962.html.
3. ISO. Information technology -- Virtualization Management Specification. [EB/OL].https://www.iso.org/standard/63962.html.
4. Vmware.Inc. Vmware. [EB/OL]. https://www.vmware.com/cn.html.
5. Volker Ruppert. Bochs, think inside the bochs. [EB/OL]. http://bochs.sourceforge.net/.
6. 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(10):1309-1319.
7. 李舟军, 张俊贤, 廖湘科,等. 软件安全漏洞检测技术[J]. 计算机学报, 2015,38(4):717-732.
8. 郭曦, 王盼. 相关路径静态分析中协同式逆向推理方法[J]. 软件学报, 2015, 26(1):1-13.
9. 甘水滔, 秦晓军, 陈左宁,等. 一种基于特征矩阵的软件脆弱性代码克隆检测方法[J]. 软件学报, 2015(2):348-363.
10. Ganapathy V, Jha S, Chandler D, et al. Buffer overrun detection using linear programming and static analysis[C]// ACM Conference on Computer and Communications Security. ACM, 2003:345-354.
11. Zhang D, Liu D, Wang W, et al. Testing C Programs for Vulnerability Using Trace-Based Symbolic Execution and Satisfiability Analysis[C]// International Conference on Dependable Systems and Networks (DSN'10). IEEE press, 2010:321-338.
12. 王伟光, 曾庆凯, 孙浩. 面向危险操作的动态符号执行方法[J]. 软件学报, 2016(5):1230-1245.
13. 崔展齐, 王林章, 李宣东. 一种目标制导的混合执行测试方法[J]. 计算机学报, 2011,34(6):953-964.
14. 李筱, 周严, 李孟宸,等. C/C++程序静态内存泄漏警报自动确认方法[J]. 软件学报, 2017, 28(4):827-844.
15. 孙浩, 李会朋, 曾庆凯. 基于信息流的整数漏洞插装和验证[J]. 软件学报, 2013, 24(12):2767-2781.
16. 邹琼, 伍鸣, 胡伟武,等. 基于插桩分析的 Java 虚拟机自适应预取优化框架[J]. 软件学报, 2008, 19(7):1581-1589.
17. 马金鑫,李舟军,张涛,沈东,章张锴. 基于执行踪迹离线索引的污点分析方法.软件学报,2017,28(9).
18. 杨智, 殷丽华, 段洣毅,等. 基于广义污点传播模型的操作系统访问控制[J]. 软件学报, 2012, 23(6):1602-1619.
19. Sang K C, Woo M, Brumley D. Program-Adaptive Mutational Fuzzing[C]// Security and Privacy. IEEE, 2015.
20. 马金鑫, 张涛, 李舟军,等. Fuzzing 过程中的若干优化方法[J]. 清华大学学报(自然科学版), 2016(5):478-483.
21. Godefroid P, Kiezun A, Levin M Y. Grammar-based whitebox fuzzing[C]// ACM Sigplan Conference on Programming Language Design and Implementation. ACM, 2008:206-215.
22. 李伟明, 于俊清, 艾少波. PyFuzzer:自动化高效内存模糊测试方法[J]. 通信学报, 2013(s2):64-68.
23. 欧阳永基, 魏强, 王清贤,等. 基于异常分布导向的智能 Fuzzing 方法[J]. 电子与信息学报, 2015, 37(1):143-149.
24. 黄亮, 冯登国, 连一峰,等. 一种基于多属性决策的 DDoS 防护措施遴选方法[J]. 软件学

报, 2015, 26(7):1742-1756.
25. 汪黎, 杨学军, 王戟,等. 操作系统内核程序函数执行上下文的自动检验[J]. 软件学报, 2007, 18(4):1056-1067.
26. 冯登国, 张敏, 张妍,等. 云计算安全研究[J]. 软件学报, 2011, 22(1):71-83.
27. 林闯,苏文博, 孟坤,等. 云计算安全:架构、机制与模型评价[J]. 计算机学报, 2013, 36(9):1765-1784.
28. 张玉清, 王晓菲, 刘雪峰,等. 云计算环境安全综述[J]. 软件学报, 2016, 27(6):1328-1348.
29. 朱民, 涂碧波, 孟丹. 虚拟化软件栈安全研究[J]. 计算机学报, 2017, 40(2):481-504.
30. Elhage N. Virtunoid: A KVM Guest → Host privilege, escalation exploits. In Black Hat USA, 2011.
31. Wang G, Estrada Z J, Pham C, et al. Hypervisor introspection: a technique for evading passive virtual machine monitoring[C]// Usenix Conference on Offensive Technologies. USENIX Association, 2015:12-12.
32. Panuccio A, Bicego M, Murino V. A Hidden Markov Model-Based Approach to Sequential Data Clustering[M]// Structural, Syntactic, and Statistical Pattern Recognition. Springer Berlin Heidelberg, 2002:734-743.
33. Mamon R S, Elliott R J, Markov AA. Hidden Markov models in finance[J]. International, 2007, 77(2):257 - 286.
34. America Fuzzy Lop. AFL[EB/OL]. http://lcamtuf.coredump.cx/afl/.
35. vger.kernel.org. Trinity[EB/OL]. http://codemonkey.org.uk/projects/trinity/.
36. Conflare. peach[EB/OL]. http://www.peachfuzzer.com/.