计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(12):1−8 [doi: 10.15888/j.cnki.csa.006683]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

恶意代码动态分析中的反虚拟化问题研究①

莫建平[1,2], 应凌云[1,2], 苏璞睿[1,2], 王嘉捷[3]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100049)
[3](中国信息安全测评中心, 北京 100085)
通讯作者: 莫建平, E-mail: mojianping15@mails.ucas.ac.cn

摘　要: 反虚拟化是当前影响恶意代码动态分析系统全面获取样本行为数据的重要因素. 本文提出从恶意代码动态分析环境的主机环境, 网络环境和用户交互环境进行系统的反虚拟化对抗方法, 并将反虚拟化对抗实现在已有的动态分析系统上, 实验结果表明反虚拟化对抗有效的增强了动态分析系统获取样本行为数据的能力.

关键词: 恶意代码分析; 动态分析; 反虚拟化

引用格式: 莫建平,应凌云,苏璞睿,王嘉捷.恶意代码动态分析中的反虚拟化问题研究.计算机系统应用,2018,27(12):1–8. http://www.c-s-a.org.cn/1003-3254/6683.html

Anti-Virtualization in Dynamic Analysis of Malicious Code

MO Jian-Ping[1,2], YING Ling-Yun[1,2], SU Pu-Rui[1,2], WANG Jia-Jie[3]

[1](Institute of Software Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: Anti-virtualization is currently an important factor affecting the overall acquisition of sample behaviour data by a dynamic analysis system of malicious code. This study proposes a systematic anti-virtualization confrontation method from host environment, network environment, and user interaction environment of dynamic analysis environment of malicious code, and implements the anti-virtualization confrontation in the existing dynamic analysis system. Experimental results show that the anti-virtualization confrontation effectively enhances the dynamic analysis system's ability to capture sample behavior data.

Key words: malicious code analysis; dynamic analysis; anti-virtualization

动态分析具有不受加壳、混淆等代码保护技术影响的优点, 是分析新恶意代码的首选方法. 动态分析通常在虚拟环境中运行样本, 利用监控模块提取样本的进程、内存、文件、注册表、网络等行为数据, 通过对这些行为数据的汇总分析来推断样本的功能和恶意性. 反虚拟化是造成动态分析系统无法获取样本全部行为数据的重要因素. 样本在检测到运行在虚拟环境后, 隐藏恶意行为或直接退出, 导致动态分析系统只能获取样本的部分行为数据, 影响动态分析的准确性. 本文将对现有的针对恶意代码动态分析的反虚拟化技术进行调研, 并提出系统的反虚拟化对抗方案, 接着将反虚拟化对抗应用到已有的基于 QEMU 的动态分析系统上, 通过实验验证了反虚拟化对抗有效性.

本文的组织结构如下, 第一节介绍现有的反虚拟

① 基金项目: 国家自然科学基金 (61502468, U1736209); “十三五”全军共用信息系统装备预研基金 (6140134040216ZK65002)
Foundation item: National Natural Science Foundation of China (61502468, U1736209); Preliminary Study Fund of Military Shared Information System Equipment of Thirteenth Five-Year Plan (6140134040216ZK65002)
收稿时间: 2018-05-08; 修改时间: 2018-06-04; 采用时间: 2018-06-19; csa 在线出版时间: 2018-12-03

© 中国科学院软件研究所 http://www.c-s-a.org.cn

化对抗方法并指出其存在的不足, 第二节对目前的反虚拟化方法进行分类阐述, 接着第三节介绍如何对抗这些反虚拟化方法, 第四节将反虚拟化对抗应用到已有的恶意代码动态分析系统, 并通过实验验证对抗的有效性, 最后第五节是全文的总结和展望.

1 反虚拟化对抗现状

针对样本的反虚拟化问题, 目前提出的解决思路主要有两种. 一是构建更难被检测, 更透明的分析系统. 如基于硬件虚拟化技术实现的动态分析系统 Ether[1], V2E[2], 基于真实硬件 (bare-metal) 的 BareBox[3], BareCloud[4]和 LO-PHIL[5]都试图通过增加分析系统的透明性对抗反虚拟化检测. 但是 Thomas Raffetseder[6]等人指出硬件虚拟化技术能被检测出来, 基于 bare-metal 的方法则代价高昂, 难以大规模应用, 并且由于缺乏对样本执行的细粒度监控, 容易受到样本的拖延技术 (样本通过睡眠, 空操作等方式使动态分析超时的技术) 的影响.

二是识别样本对虚拟环境的检测手段并绕过检测. 如 Cobra[7]使用二进制翻译来监控每个基本块的执行, 通过替换掉所有可能的针对 Cobra 的检测指令来绕过样本反虚拟化检查, 但是 Cobra 使用细粒度的二进制插桩, 性能下降明显. 而 Balzarotti[8]等人在虚拟分析环境上重放参考主机 (物理主机) 上的执行记录, 通过比较样本在两次执行中的差异来发现样本对虚拟环境的检测方法.

动态分析环境包括主机环境, 网络环境和用户交互环境三个方面, 仅仅考虑主机环境的反虚拟化对抗是不够的, 样本可以通过对网络环境或用户交互的探测进行反虚拟化.

2 反虚拟化方法概述

样本的反虚拟化基于模拟分析环境和真实环境的差异进行, 这样的差异可以分为主机环境差异, 网络环境差异和用户交互环境差异三个方面.

2.1 基于主机环境差异的反虚拟化方法

动态分析系统主机环境的构建主要有基于硬件模拟器 (如 Bochs, QEMU), 基于硬件虚拟化 (如 VirtualBox, VMware), 基于虚拟机监视器 (又称 hypervisor, 如 Xen, VMware ESXi) 和基于真实硬件四类. 现有针对主机环境差异的虚拟化检测方法主要包括基于硬件特征的检测方法和基于系统特征的检测方法两类.

硬件特征主要包括 CPU、内存、硬盘驱动器、CD/DVD 驱动器、BIOS、ACPI、显卡和网卡几类. 表 1 对这些检测方法进行了总结. 表中列出了针对每类特征的各个检测项并举例说明, 注释部分对检测项或例子做出了说明. 比如针对内存大小的检测项, 动态分析系统由于考虑到资源的消耗和分析的并行度, 一般会分配给虚拟机较小的内存, 如 1 GB.

表 1 基于硬件特征的反虚拟化方法

检测类别	检测项	例子	注释
CPU	核数/个数	<2	市面 CPU 往往多个或多核
	Vendor	QEMU Vritual CPU version 2.5+	使用 Cpuid with eax=0 获得
	Hypervisor version	VBoxVBoxVBox; VMwareVMware	使用 Cpuid with eax=0x40000000 获得
	Hypervisor bit	为 1	使用 Cpuid with eax=1 获得
	注册表键值	QEMU Vritual CPU version 2.5+	ProcessorNameString
内存	大小	内存小于 2 GB	市面电脑内存多在 2 GB 以上
硬盘	大小	硬盘小于 60 GB	市面电脑硬盘多在 320 GB 以上
	注册表键值	DiskVMware; VBOX HARDDISK	ControlSet001\Enum\IDE\; Logical Unit Id 0\Identifier (Scsi Port 0)
CD/DVD	注册表键值	CDRomQEMU; VBOX CD-ROM	ControlSet001\Enum\IDE\; Logical Unit Id 0\Identifier (Scsi Port 1)
BIOS	注册表键值	Bochs -1; VMware, Inc; VirtualBox -1	System\SystemBiosVersion; BIOS\SystemManufacturer; BIOS\BIOSVendor
	Raw SMBIOS 中的特征字符串	SeaBIOS; VMware, Inc	从 csrss.exe 进程的 0xE0000~0xF0000 dump 得到
ACPI	注册表键值	BOCHS_VBOX__	HARDWARE\DSDT\BOCHS_\; HARDWARE\DSDT\VBOX__\
显卡	Raw 显卡 BIOS 中的特征字符串	VMware SVGA; VboxVgaBios	从 csrss.exe 进程的 0xE0000~0xF0000 dump 得到

© 中国科学院软件研究所 http://www.c-s-a.org.cn

CPU 个数和核数, 内存大小, 硬盘大小等特征为通用的反虚拟化特征, 对所有类型的虚拟化平台都适用. 其它硬件特征则是虚拟化平台相关的, 不同的虚拟化平台含有不同的特征值, 同一虚拟化平台的不同版本的特征值可能也不一样. 基于真实硬件的主机环境由于使用真实的硬件, 并且一般 CPU 为多个或为多核, 内存和硬盘空间比较大, 因而不包含用于反虚拟化的硬件特征.

基于系统特征的检测方法可以分为两类: 一是查找系统进程列表, 文件, 注册表, 服务列表等是否含有模拟器相关的特征字符串. 二是基于时间的检测方法, 包括检测系统启动时间, 通过比较在真实系统和虚拟机上执行特定代码或任务的绝对或相对时间差来探测虚拟机的存在.

虚拟化平台为了实现虚拟化的功能或增加平台的易用性, 需要向 guest 引入必要的组件, 如设备驱动, DLL 文件, EXE 文件等, 可能还会注册随系统启动的服务或程序, 这些组件的信息会在系统安装或运行过程中保存到注册表或日志文件中. 虚拟化平台额外引入的这些文件成为反虚拟化利用的特征. 这些特征是虚拟化平台相关的. 表 2 总结了基于系统特征的反虚拟方法. 其中进程和服务特征可以通过遍历进程列表和服务列表进行检查.

表 2 基于系统特征的反虚拟化方法

检测类别	检测项	例子	注释
进程	进程名称	vmtoolsd.exe; VBoxTray.exe	额外引入的进程
服务	服务名称	VMware Physical Disk Helper Service; VirtualBox Guest Additions Service	额外引入的服务
文件	文件内容中的特征字符串	setuplog.txt; vmmouse.sys; VBoxVideos.sys; VBoxControl.exe	驱动, DLL, EXE, 日志文件等
注册表	虚拟化平台相关的键和值项	HARDWARE\DSDT\VBOX__\; ControlSet001\Services\VBoxGuest	硬件特征相关;额外引入软件相关
系统启动时间	系统启动时间	<12 分钟	镜像快照拍摄时间较早, 通过 GetTickCount 获得
性能差异	绝对或相对时间差	Cpuid; GetProcessHeap/CloseHandle	rdtsc; GetTickCount 等 API 测量代码执行时间

硬件模拟器需要模拟所有的硬件操作, 还需要将 guest CPU 的代码动态翻译成宿主机 CPU 的代码, 基于硬件虚拟化或 hypervisor 的虚拟化平台虽然减轻了在硬件操作上的模拟压力, 但在性能上还是比相同配置的物理主机要低, 这样的性能差异通过可以测量虚拟机和物理主机运行特定代码或任务耗时绝对时间差或相对时间差体现出来. 绝对时间差是指运行同一代码或任务, 虚拟机和物理机耗时的差值或比值, 而相对时间差是指两段不同代码或两个不同任务在虚拟机和物理机上耗时的差值或比值之间的差异. 为了增加测量的稳定性, 通常会循环执行特定代码或任务多次, 取平均值作为测量结果. 时间的测定使用 rdtsc 指令, GetLocalTime, GetTickCount, QueryPerformance-Counter 和 QueryPerformanceFrequency 等 API. 特定代码包括 cpuid 指令, GetProcessHeap 和 CloseHandle 及其它在虚拟机和物理机上执行时间有显著差异的代码或任务. 基于时间的检测方法为通用的反虚拟化方法, 适用于所有虚拟化平台.

2.2 基于网络环境差异的反虚拟化方法

真实网络的站点和服务程序数量巨大, 而模拟网络在实现上往往只模拟一个程序和一个站点, 这样的矛盾性导致了基于网络环境差异的反虚拟化方法. 模拟网络和真实网络的差异可以归纳为如下两个方面: 1) 性能差异, 模拟网络通常与分析主机处于同一局域网, 对请求响应迅速, 延时比真实网络小, 2) 服务细节差异, 模拟服务往往响应固定模式的消息, 和真实服务在细节方面存在差异, 如 DNS 查询总是响应相同的 IP, HTTP 对不同动态参数的 url 响应相同页面, 不同 HTTPS 站点使用相同的证书等. 表 3 总结了针对常见协议的反虚拟化方法. 其中性能差异的数据表示模拟网络服务的响应速度, 括号中数据为真实服务的响应速度, 测量的是服务成功连接后, 一次请求的响应时间. 选用国内常见站点作为测试目标.

2.3 基于用户交互环境差异的反虚拟方法

恶意代码动态分析系统为了提高分析效率, 往往

© 中国科学院软件研究所 http://www.c-s-a.org.cn

采用自动化分析, 没有用户使用系统, 因而与有用户使用的系统存在明显差异. 基于用户交互环境差异的检测方法可以分为用户使用记录和用户交互动作两类.

表 3 常见协议的反虚拟化方法

协议	性能差异	服务细节差异
DNS	<10 ms (45.7 ms)	不同域名响应相同 IP, PTR 选项不受支持等
HTTP	<10 ms (36.4 ms)	动态参数响应相同结果, 不支持重定向, 不同站点具有相同程序版本
HTTPS	<10 ms (67.7 ms)	不同站点具有相同的证书, 程序版本
FTP	<5 ms (29.8 ms)	不同站点具有相同的程序版本和旗标信息
SMTP	<5 ms (21.4 ms)	不同站点具有相同的服务器名和旗标信息
POP3	<5 ms (26.1 ms)	不同站点具有相同的服务器名和旗标信息

用户使用记录是指用户在使用操作系统时留下的用户痕迹, 包括如下几点: 剪切板是否有临时数据; 常见的软件是否安装; 是否有一定量的个人数据, 图片, 文档, 音视频等; 是否有近期的浏览器历史记录; 是否具有一定量的用户登录记录, 系统日志等.

用户交互动作主要指鼠标的移动点击和键盘的击键. 键盘和鼠标是 PC 机的主要输入设备, 用户使用系统时会经常进行鼠标和键盘的操作, 而动态分析系统由于是自动化分析, 故缺乏相关的用户交互动作. 鼠标的位置可以使用 GetCursorPos API 获取, 键盘击键的可以通过安装全局 hook 进行监视. 表 4 总结了基于用户交互环境差异的反虚拟化方法.

表 4 基于用户交互环境差异的反虚拟化方法

检测类别	检测项	注解
用户使用记录	剪切板是否有数据	GetClipboardData
	常用软件是否安装	Office, PDF, RAR, JDK 等
	是否有个人数据	图片, 音视频, 文档等
	浏览器历史记录	IE, Firefox, Chrome 等
	系统日志	应用异常日志, 硬件错误日志等
用户交互动作	鼠标移动, 点击	GetCursorPos, SendMessage 等
	键盘击键	SendMessage, SetWindowsHookEx 等

3 反虚拟化对抗

3.1 对抗基于主机环境差异的反虚拟化方法

虽然部分特征可以通过定制镜像或 inbox 的 API hook 应对, 但是并不彻底. 比如注册表中硬盘相关的字符串, 可以通过修改注册表键值移除, 但是系统重启之后, 原有的键值由于备份的原因仍然会出现, 而通过 inbox 的 hook 注册表键值打开, 查询等 API 的方式隐藏相关字符串方式, 难免对操作系统做出修改, 使得隐藏本身引入了新的检测虚拟环境的特征. 由于 QEMU 在恶意代码动态分析中被广泛使用, 并且是开源的硬件模拟器, 便于从源码移除字符串类型的特征, 实现 outbox 的 API hook 隐藏系统信息, 因此本文选取在 QEMU 实现对抗基于主机环境的反虚拟化方法.

从源码移除硬件特征需要熟悉 QEMU 启动流程, 通过搜索得到所有可能的硬件特征字符串, 然后利用静态数据流分析, 判断可疑字符串是否为模拟硬件使用, 并评估是否可移除或替换, 最后重新编译, 测试修改是否有效及是否对系统运行产生负面影响.

对于不便从源码移除的特征, 如内存大小, hypervisor bit 及基于性能差异的反虚拟化方法, 采用 outbox 的 API hook 进行对抗. Outbox 的 API hook 框架如图 1 所示. 首先获取被监控目标, 每当遇到一个新的进程时, 通过进程的环境控制块 TEB 获取进程名, 与样本名比较, 若一致则表明该进程为被监控目标, 提取进程控制块 ETHREAD 地址和当前 CR3 为进程标志. 然后利用得到的 CR3 和 ETHREAD 对轮转的进程/线程进行过滤, 当命中目标进程且当前 IP 地址为需要监控的 API 地址时, 接管当前进程控制流, 进行 hook 处理. 最后重置相关参数, 进行清理工作, 为下一轮 hook 做准备.

API hook 的实现需要注意如下问题: (1) 需要对被监控进程/线程及其子进程/子线程进行; (2) 避免对虚拟机运行产生负面影响; (3) 尽量减少 hook 带来的性能损耗. 为了监控目标进程派生的线程树和进程树, 对进程/线程的创建和销毁的 API 进行监控, 使用列表维护目标进程派生的后代, 在进程过滤时查找该列表. 为了避免对虚拟机运行产生负面影响, API 监控只针对目标进程及其后代进程和线程. 而为了降低 hook 带来的性能消耗, 在过滤进程时, 若当前 IP 处在用户空间, 则认为进程没有切换, 不再查找当前进程的 ETHREAD 地址进行进程过滤. 在首次寻找目标进程时, 可以维持一个列表用于缓存已经检视过的进程 ETHREAD 和 CR3, 以加速目标进程的匹配. 目前的实现没有对目标进程的子进程树和子线程树进行监控.

表 5 中总结了基于硬件特征的反虚拟化对抗方法. 根据表 2, 针对基于系统特征的反虚拟化方法主要包括 6 类. 其中进程列表和服务列表不包含 QEMU 相关特征字符串, 无需对抗. 而注册表键值和特殊文件所包含的特征字符串全部是硬件相关的, 经过测试, 在从源

© 中国科学院软件研究所 http://www.c-s-a.org.cn

码移除硬件特征字符串后, 注册表和特殊文件 (使用新的系统镜像) 不再包含 QEMU 相关特征字符串.

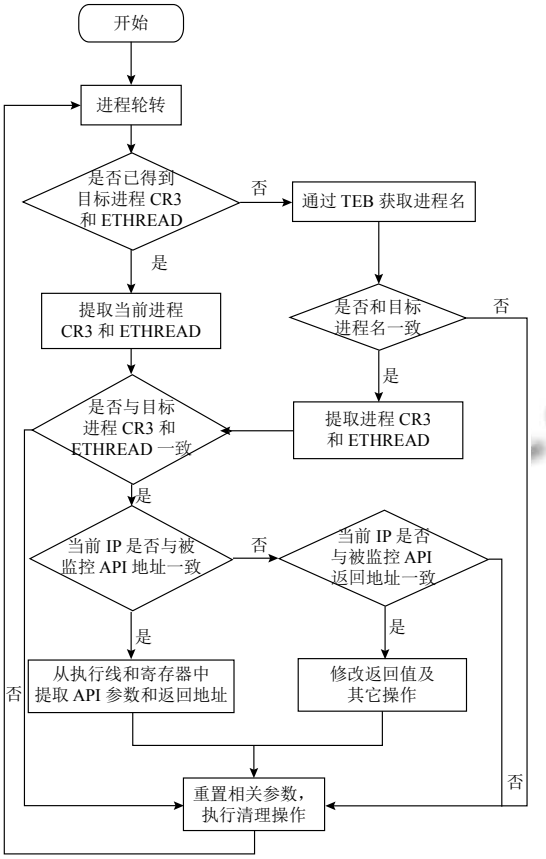

图 1 Outbox 的 API hook 流程图

表 5 基于硬件特征的反虚拟化对抗方法

检测类别	检测项	对抗方法
CPU	核数/个数	命令行选项-smp 指定多个多核 CPU
	Vendor	源码移除特征字符串
	Hypervisor version	源码移除特征字符串
	Hypervisor bit	Hook cpuid 指令, 重置 hypervisor bit
	注册表键值	源码移除特征字符串
内存	大小	Hook GlobalMemoryStatus 等 API
硬盘	大小	制作硬盘空间更大的系统镜像
	注册表键值	源码移除特征字符串
CD/DVD	注册表键值	源码移除特征字符串
BIOS	注册表键值	源码移除特征字符串
	Raw SMBIOS 中的特征字符串	SeaBIOS 源码移除特征字符串
ACPI	注册表键值	源码移除特征字符串
显卡	Raw 显卡 BIOS 中的特征字符串	SeaVGABIOS 源码移除特征字符串
网卡	MAC 地址前缀	命令行选项-net nic 指定随机 MAC 地址

针对基于时间的检测方法, 没有通用有效的对抗方案. 虽然随着版本的更新迭代, 硬件模拟器的性能在不断提升, 但目前仍然存在某些特殊任务能在宏观或微观上区分模拟硬件和真实硬件. 如使用 GPU 渲染高帧率的游戏画面, 在模拟硬件上耗时远超过真实硬件. Windows XP 上 GetProcessHeap 与 CloseHandle API 调用耗时的比值在模拟硬件上超过 10 倍, 而在真实硬件上两者耗时相近. 对于特征明显, 并且使用特定指令或 API 进行时间测量的任务, 可以通过 hook 相关 API, 修改时间测量结果的方法对抗. 但是对于样本运行某些在真实硬件耗时很少, 而虚拟硬件耗时较多的"正常"的任务, 从而使动态分析超时的拖延战术, 在耗时任务不易识别的情况下, 很难实施有效的对抗方法.

时间测量计算的是代码执行前后时间的差值, 因此 hook API 时关注的不是函数当前的返回值, 而是前后两次返回值的差值. 简单的将前后两次的差值置为定值或定值加上一个小范围的随机数, 不能满足对抗的需要. 差值为定值容易被检测出来, 且不符合现实情况, 因为操作系统任务调度和外部事件的不确定性, 即使相同代码, 执行时间也可能不一致. 加上随机值模拟了执行时间的不确定性, 但仍可以通过测量 Sleep 的执行时间进行检测. 如果测量得到的差值远小于 Sleep 指定的时间, 则说明 Sleep 函数或时间测量函数被 patch, 这都指示了样本运行在虚拟机中. 因此本文提出了如图 2 所示的差值修改方法.

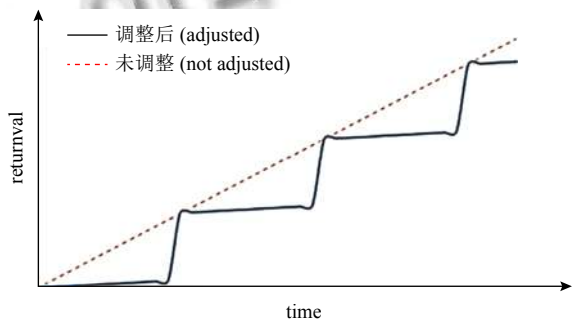

图 2 时间测量函数返回值曲线

图中虚线为未调整的返回值曲线, 斜率为 1, 实线为调整后的返回值曲线. 实线形状类似向上的台阶, 台阶平面波动向上, 即差值随机化. 台阶向上跃迁即不做修改, 直接返回函数调用结果. 跃迁的条件是自上次测量函数的调用经过了超过阈值的时间, 或者这期间调用了 Sleep, SleepEx 和 NtDelayExecution 等延时函数,

© 中国科学院软件研究所 http://www.c-s-a.org.cn

跃迁可以对抗通过测量 Sleep 执行时间检测虚拟机的方法, 减小对两次差值测量之间的代码执行时间产生的影响. 阈值的大小决定了隐藏性能差异的大小. 样本第一次调用时间测量函数时, 不对返回值进行修改. 返回值修改只针对被监控进程进行, 不影响系统中其它进程对时间测量函数的调用. 差值修改方法虽然在 QEMU 上实现, 但对其它虚拟化平台也适用. 表 6 总结了针对 QEMU 的基于系统特征的反虚拟化对抗方法.

表 6 基于系统特征的反虚拟化对抗方法

检测类别	检测项	对抗方法
进程	进程名称	无需对抗
服务	服务名称	无需对抗
文件	文件内容中特征字符串	移除硬件特征后, 无需对抗
注册表	虚拟化平台相关的键和值项	移除硬件特征后, 无需对抗
系统启动时间	系统启动时间	Hook GetTickCount, 随机化系统启动时间
时间差	绝对或相对时间差	Hook 时间测量指令和函数, 修改返回值为类似梯形的曲线

3.2 对抗基于网络环境差异的反虚拟化方法

INetSim[9] (Internet Service Simulation Suite) 是一款由 Perl 编写的开源网络服务模拟软件. 其提供 DNS, HTTP, SMTP, IRC 等多种常见的网络服务模拟, 以支持恶意软件的动态分析, 同时支持使用配置文件对模拟服务进行个性化配置. 因此本文在 INetSim 上实现基于网络环境差异的反虚拟化方法的对抗.

基于网络环境差异的反虚拟化方法包括性能差异和服务细节差异. 针对性能差异, 在服务响应请求时延时一定时间, 如 20 ms 至 40 ms, 以与正常网络的延时相近. 由于 INetSim 采取 Fork 模式, 每个请求都 Fork 一个子进程进行处理, 因此使用进程睡眠延时的方式虽然降低了服务的吞吐率, 但是不会对后续同类型和和其它类型请求造成影响.

针对服务细节差异, 主要采用随机化的方法改变原来服务返回的固定信息进行对抗, 如 DNS 服务在解析不同域名时, 从指定 IP 池中随机返回一个, 在服务初始化时, 从指定的程序版本中随机返回一个等. 由于网络服务独立于主机环境, 实现在 INetSim 的对抗方法适用于所有虚拟化平台. 表 7 总结了针对常用协议的反虚拟化对抗方法.

表 7 常用协议的反虚拟化对抗方法

协议	性能差异	服务细节差异
DNS	随机延时	随机 IP 池, 增加 PTR 支持
HTTP	随机延时	增加重定向, 动态参数匹配, 随机化程序版本
HTTPS	随机延时	动态证书, 随机化程序版本
FTP	随机延时	随机化程序版本和旗标信息
SMTP	随机延时	随机化服务器名和旗标信息
POP3	随机延时	随机化服务器名和旗标信息息

3.3 对抗基于用户交互环境差异的反虚拟化方法

根据表 4, 基于用户交互环境差异的反虚拟化方法主要包括用户使用记录和用户交互动作两类. 针对用户交互记录, 主要通过定制镜像的方法进行对抗. 由于组内恶意代码动态分析系统已有的系统镜像经过了几年的定制更新, 常见的软件, 用户数据, 浏览器记录, 系统登录日志等用户使用记录都具备, 满足对抗要求. 考虑到从新的系统镜像重新进行一轮定制的时间代价和最终效果, 本文直接使用这个系统镜像对抗基于用户使用记录的反虚拟化方法.

使用脚本模拟鼠标的移动点击和键盘的击键, 属于 inbox 的方式, 在对抗的同时也引入了新的反虚拟化特征. 因此本文提出通过发送硬件事件的 outbox 方式提供用户鼠标和键盘动作的模拟. 真实用户操作在基于 QEMU 的虚拟机上, 都是以发送事件的方式改变鼠标和键盘的状态. 本文实现了 3 种发送硬件事件的方式, 分别是基于 QEMU 主循环的, 基于线程的和基于 QEMU 定时器的. QEMU 使用一个大循环处理外部事件和中断, 基于主循环的方式即在主循环中插入发送事件的代码, 但由于主循环执行的频率较高, 而正常用户的操作速度较慢, 因此需要一个计数器指定主循环执行若干次后触发硬件发送事件, 但是使用循环次数不好估计时间, 难以精确事件发送的间隔. 基于线程的方式则在 QEMU 启动时单独创建一个线程, 线程在随机睡眠一段时间后发送硬件事件, 为了便于控制, 模拟鼠标和键盘的操作至少需要两个线程. QEMU 时钟支持定时器以定期完成某些任务, 每次主循环都会检测是否有到期的定时器, 如果有, 则执行定时器预设的函数. 基于定时器的方式在 QEMU 时钟上安装定时器, 定时器注册的函数发送硬件事件并重置定时器, 该函数的执行时间不能过长, 否则影响对其它外部事件的响应. 考虑到三种实现方式的侵入性和性能损耗, 采用基于定时器的实现鼠标随机移动和点击, 采用基于线程的方式实现键盘的随机击键. 表 7 总结了基于用户交互环境差异的反虚拟化对抗方法.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 8 基于用户交互环境差异的反虚拟化对抗方法

检测类别	检测项	对抗方法
用户使用记录	剪切板是否有数据 常用软件是否安装 是否有个人数据 浏览器历史记录 系统日志	使用组内已有的定制过的镜像对抗
用户交互动作	鼠标移动, 点击 键盘击键	发送硬件事件的方式随机移动鼠标 发送硬件事件的方式随机敲击键盘

4 实验与分析

4.1 实验方案

实验使用的样本包括两个部分, 来源为网络收集, 组内已有样本和作者编写的样本. 样本集一的分布如表 9 示, 总计 113 个. 其中包括 2 个开源反虚拟化软件 pafish[10]和 al-khaser[11], 它们包含了多项针对主机环境差异和用户交互环境差异的反虚拟检测技术. 除了针对 QEMU, VMware, VBOX 等常见虚拟化平的的反虚拟化样本外, 还包括 31 个自编写样本, 根据第 2 节总结的反虚拟方法, 为每个检测项编写对应的反虚拟化样本作为补充, 同时为了使样本具有"恶意功能", 样本在相应的反虚拟检测通过后, 会表现出若干敏感行为, 包括设置自身为开机自启动, 访问 baidu.com 获取当前时间, 启动记事本进程并在 5 秒后将其关闭及在系统目录创建名为 invasion.txt 的空文件. 样本集二为随机选择的 1274 个样本, 用于测试改进在真实样本上的效果.

表 9 反虚拟化对抗实验样本分布

类别	数量	注释
开源反虚拟软件	2	pafish 和 al-khaser, 包含多类检测技术
基于网络环境特征	11	作者编写, 实现表 3 的反虚拟方法
基于主机环境特征	18	作者编写, 实现表 1 和表 2 的反虚拟方法
基于用户交互特征	2	作者编写, 检测鼠标和键盘操作的反虚拟化方法
QEMU	20	开源虚拟化软件
VMware	30	商用虚拟化软件
VBOX	30	开源虚拟化软件
合计	113	

如果改进后的动态分析系统观察到更多的样本行为, 则认为改进是有效的. 样本行为数目统计样本的进程操作, 网络访问, 文件操作和注册表操作的数量. 针对样本可能在不同的运行过程中访问不同站点, 创建不同的文件名的临时文件等影响样本行为条目的随机性问题, 在统计相同类型行为数量时, 只关注行为操作对象的相异性, 而不关注操作对象的具体值, 如样本在一次运行过程中访问 A, B, C 3 个站点, 在另一次运行过程中访问了 A, B, D 3 个站点, 则认为样本两次运行产生的网络行为数目相同, 都访问了 3 个不同的站点.

本文将反虚拟化对抗应用到组内已有的基于 QEMU 的恶意代码动态分析系统上, 并进行改进前后对比实验. 实验中每个样本的分析时间为默认的 2 分钟, 虚拟机镜像为 Windows XP 32 位.

4.2 实验结果

将样本集上传到改进前和改进后的动态分析系统, 然后下载 XML 格式的分析报告和网络数据包 cap 文件, 统计样本的行为数目, 进行对比分析. 表 10 汇总了开源反虚拟化工具 pafish 和 al-khaser 在改进前后针对 QEMU 的检测项的通过情况, 其中"X"表示未通过, "√"表示通过. 可以看到, 经过反虚拟化对抗, 动态分析系统通过了 pafish 和 al-khaser 所有针对 QEMU 的反虚拟化检测项.

表 10 pafish 和 al-khaser 实验结果

样本	检测项	改进前	改进后
al-khaser	Number of processors(<2)	X	√
	Number of cores(<2, WMI)	X	√
	CPU hypervisor bit	X	√
	CPU vendor	X	√
	Disk size(<80 G, SetupDi_diskdriver)	X	√
	Disk size(<80 G, DeviceIoControl)	X	√
	Disk size(<80 G, GetDiskSpaceEx)	X	√
	Disk size(<80 G, WMI)	X	√
	Memory size(<1 G, GlobalMemoryStatusEx)	X	√
	Mouse movement	X	√
	RDTSC Locky trick(CloseHandle/GetProcessHeap)	X	√
	RDTSC(force VM exit)	X	√
pafish	CPU vendor	X	√
	RDTSC	X	√
	RDTSC (force VM exit)	X	√
	Hypervisor bit	X	√
	Hypervisor vendor	X	√
	Mouse movement	X	√
	Disk size(<60 G, DeviceIoControl)	X	√
	Disk size(<60 G, GetDiskFreeSpaceEx)	X	√
	NumberOfProcessors(<2, raw access)	X	√
	NumberOfProcessors(<2, GetSystemInfo)	X	√
	Physical memory(<1 G)	X	√
	System uptime(<12 min, GetTickCount)	X	√
	CPUID(CPU brand string 'QEMU VirtualCPU')	X	√
	Disk(Scsi port->bus->target id->logical unit id->0 identifier)	X	√
	Reg(HKLM\HARDWARE\Description\System "SystemBiosVersion")	X	√

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 11 展示了所有反虚拟化样本的实验结果 (不计 pafish 和 al-khaser). 可见, 改进后的动态分析系统在所有针对 QEMU 的作者编写样本和收集样本中都观察到了更多行为. 而针对 VMware 和 VBOX 的反虚拟化样本则和改进前的分析系统表现相同, 这是因为改进前后的动态分析系统都是基于 QEMU 的, 针对 VMware 和 VBOX 的反虚拟化方法自然不起作用, 因此改进后的动态分析系统并没有观察到这些样本的更多行为.

在样本集二中, 378 个样本观察到更多行为数据, 为总样本数的 29.7%. . 说明反虚拟化对抗使动态分析系统更全面的了解样本行为, 同时也说明反虚拟化在实际样本中使用广泛. 综合样本集一和样本集二的实验结果, 反虚拟化对抗显著增强了动态分析系统获取样本行为数据的能力.

表 11 反虚拟化对抗实验结果 (不计 pafish 和 al-khaser)

类别	观察到更多行为的样本数目	样本总数
基于网络环境特征	11	11
基于主机环境特征	18	18
基于用户交互特征	2	2
QEMU	20	20
VMware	0	30
VBOX	0	30
合计	53	113

5 结论和展望

本文对当前针对恶意代码动态分析系统的反虚拟化方法进行了分类总结, 提出从恶意代码动态分析系统的主机环境, 网络环境和用户交互环境三个方面进行系统的反虚拟对抗的方法, 并将反虚拟化对抗应用到已有的基于 QEMU 的动态分析系统上, 实验结果表明反虚拟化对抗有效地帮助了动态分析系统对样本行为的全面了解. 现代 CPU 功能强大, 硬件逻辑复杂, 又缺乏公开的指令设计文档, 模拟 CPU 的指令执行效果难免会有与真实 CPU 出入的地方, 从而成为恶意代码检测虚拟 CPU 的特征. 本文目前没有对基于 CPU 语义差异的反虚拟方法进行对抗, 这是未来工作方向之一.

参考文献

1 Dinaburg A, Royal P, Sharif M, *et al*. Ether: Malware analysis via hardware virtualization extensions. Proceedings of the 15th ACM Conference on Computer and Communications Security. Alexandria, VA, USA. 2008. 51–62.

2 Yan LK, Jayachandra M, Zhang M, *et al*. V2E: Combining hardware virtualization and softwareemulation for transparent and extensible malware analysis. ACM Sigplan Notices, 2012, 47(7): 227–238. [doi: 10.1145/2365864]

3 Kirat D, Vigna G, Kruegel C. Barebox: Efficient malware analysis on bare-metal. Proceedings of the 27th Annual Computer Security Applications Conference. Orlando, FL, USA. 2011. 403–412.

4 Kirat D, Vigna G, Kruegel C. BareCloud: Bare-metal analysis-based evasive malware detection. Proceedings of the 23rd USENIX Conference on Security Symposium. San Diego, CA, USA. 2014. 287–301.

5 Wressnegger C, Yamaguchi F, Arp D, *et al*. Comprehensive analysis and detection of flash-based malware. Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. San Sebastián, Spain. 2016. 101–121.

6 Raffetseder T, Kruegel C, Kirda E. Detecting system emulators. Proceedings of the 10th International Conference on Information Security. Valparaíso, Chile. 2007. 1–18.

7 Vasudevan A, Yerraballi R. Cobra: Fine-grained malware analysis using stealth localized-executions. Proceedings of 2006 IEEE Symposium on Security and Privacy. Berkeley/Oakland, CA, USA. 2006. 264–279.

8 Balzarotti D, Cova M, Karlberger C, *et al*. Efficient detection of split personalities in malware. Proceedings of the 17th Annual Network and Distributed System Security Symposium. San Diego, CA, USA. 2010.

9 INetSim. http://www.inetsim.org. [2017-12-09].

10 Pafish. https://github.com/a0rtega/pafish. [2017-12-19].

11 Al-khaser. https://github.com/LordNoteworthy/al-khaser. [2017-12-19].

© 中国科学院软件研究所 http://www.c-s-a.org.cn