基于 angr 的对抗恶意代码沙箱检测方法的研究

张君涛 王轶骏 薛 质
(上海交通大学电子信息与电气工程学院 上海 200240)

摘 要 恶意代码在运行时会采取多种方法探测沙箱环境,从而避免自身恶意行为暴露。通过对常见沙箱检测方法的研究,在具有动态符号执行功能的开源二进制代码分析框架 angr(Advance Next Generation Research into binary analysis)的基础上,使用 Win32 API 函数挂钩、VEX 指令修补以及内存结构完善三种方法对抗沙箱检测机制。原型系统上的测试表明,该方法能够绕过常见的沙箱检测机制,同时显著优于现有的恶意代码动态分析工具。

关键词 恶意代码 对抗沙箱检测 angr 动态符号执行

中图分类号 TP3 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.02.054

ANGR-BASED DETECTION METHOD AGAINST MALICIOUS CODE SANDBOX

Zhang Juntao Wang Yijun Xue Zhi
(*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*)

Abstract Malicious code can detect sandbox environment in many ways when it runs, so as to avoid exposing its own malicious behavior. Based on the open source binary code analysis framework angr with dynamic symbol execution function, the sandbox detection mechanism was resisted by three methods: Win32 API function hook, VEX instruction repair and memory structure improvement through the research of the sandbox detection method. Experiments on the prototype system show that the proposed method can bypass the common sandbox detection mechanism and significantly outperform current dynamic analysis methods.

Keywords Malicious code Against sandbox detection angr Dynamic symbol execution

0 引 言

远控木马、僵尸网络、蠕虫以及近些年爆发的勒索软件、挖矿木马等各类恶意代码影响着用户对计算机的正常使用。针对特定目标的 APT 攻击,采用高度定制化的恶意代码进行鱼叉攻击,成为了国家信息安全面临的重大威胁。因而,恶意代码一直受到网络安全从业人员以及研究人员的广泛关注。

十多年前,研究者就已经采用沙箱技术来分析恶意代码。布谷鸟(Cuckoo)是一款开源且使用广泛的沙箱软件,利用虚拟化技术构造出与真实环境隔离且拥有快速还原能力的系统,研究人员通过分析代码在沙箱中的所有行为,进而判断是否为恶意代码。因此,针对恶意软件的分析沙箱被认为是防御 APT 的最后一道防线。病毒检测厂商比如 CrowdStrike、Malwr 以及国内的微步科技都利用沙箱技术提供恶意代码在线检测的服务。

恶意代码开发者为了逃避沙箱检测,防止自身暴露,会在恶意代码中加入环境探测代码,当发现自身运行于异常环境中时,就会停止恶意行为,运行无害代码或者结束运行,达到逃避检测的目的。据 McAfee 2017 年发布的《威胁报告》显示,反沙箱机制超过反调试、反监视、代码混淆等方法,成为恶意代码使用最为广泛的逃避检测手段[1]。此外,Lastline 实验室在 2015 年指出,具有逃避检测功能的恶意代码的比例从 2014 年年初的 35% 上升至 2014 年 12 月的 80%,并且往往结合数 10 种方法[2]。

收稿日期:2018-06-24。国家重点研发计划项目"网络空间安全"重点专项(2017YFB0803200)。**张君涛**,硕士生,主研领域:网络空间安全。**王轶骏**,讲师。**薛质**,教授。

本文采用 angr 对恶意代码进行动态分析。angr 是一款基于 Python 实现,可利用动态符号执行进行二进制代码分析的调试器[3]。相比于现有的分析方法,动态符号执行可以遍历程序不同的分支,获得恶意代码更多的行为特征。这能有效避免外部环境比如 C&C 服务器的失效对分析恶意代码产生的影响[4]。相对的,反沙箱机制却会导致符号执行面临路径爆炸的难题,特别是当符号值作为循环语句的参数时,产生大量无害分支,导致难以发现真正执行恶意行为的分支[5]。

因而本文将首先对使用最为广泛的恶意代码沙箱检测机制进行分析;接着说明我们为 angr 制定的拓展程序,用于对抗不同类型的反沙箱机制以及原型系统的实现方法;最后通过对沙箱检测软件 al-khaser 以及 paranoid fish 的测试,证明我们开发的原型系统可以绕过常见的反沙箱机制,有效防止符号执行的路径爆炸问题。

1　沙箱检测常用技术

在恶意代码分析领域,Sandboxie、布谷鸟、Anubis 等是常见的沙箱软件。然而,沙箱与真实环境存在一定的差别,沙箱为了监控恶意代码的行为、阻止恶意代码从沙盒逃逸,部分采用进程注入、函数挂钩等机制监控恶意代码的行为[6-8]。有些沙箱利用虚拟化技术构建真实操作系统,但这些虚拟机存在特定的指纹信息[9-11]。因而,恶意代码反沙箱方法可以分为基于沙箱的检测以及基于虚拟机的检测。

1.1　基于沙箱的检测

Sandboxie 与被分析的恶意代码同处一个操作系统内,为了避免恶意代码对真实系统产生影响,沙箱内的程序对真实系统的文件、注册表、进程的访问都需要获得 Sandboxie 的授权。Sandboxie 通过在内核驱动上挂载 SbieDrv. sys 实现对系统服务描述符表(SSDT)、Shadow SSDT 表的挂钩。通过将 SbideDll. dll 注入到进入沙盒的程序中,实现对所有用户态 API 函数的挂钩。因而恶意代码通过检测程序自身是否被注入 SbideDll. dll,即可成功绕过 Sandboxie 沙箱的检测。

布谷鸟沙盒是蜜网社区 2010 年谷歌编程之夏的开源项目,经过数年的开发,其功能与性能不断升级,成为了目前较为流行的沙箱系统。布谷鸟沙箱可分为主从节点,主节点主要负责任务的分发以及对恶意代码运行结果的分析展示。从节点为恶意代码的动态运行提供真实的系统环境,为了能够实现系统的快速还原,从节点一般为虚拟机。布谷鸟同样采取 dll 注入的方式来监控 Windows API 的调用。此外,布谷鸟还采用 API 挂钩的方法,因而 David 编写的 anti-cuckoo 通过检测 ntdll. dll 中的某几个关键函数是否被篡改来探测布谷鸟程序。此外,anti-cuckoo 通过检查进程的内存空间中是否存在敏感词,比如 cuckoomon、HookHandle、retaddr-check 等来探测是否运行于布谷鸟中。

1.2　基于虚拟机的检测

由于虚拟机具有快速还原的特点且与真实系统隔离,不仅仅是上一节叙述的布谷鸟、Anubis 等沙箱使用虚拟机作为恶意代码分析的宿主机,反病毒研究员在日常分析恶意代码行为时,也大都使用虚拟机作为分析系统。常用的虚拟机有 VirtualBox、Vmware、KVM、Xen 等。恶意代码针对虚拟机的检测如图 1 所示。

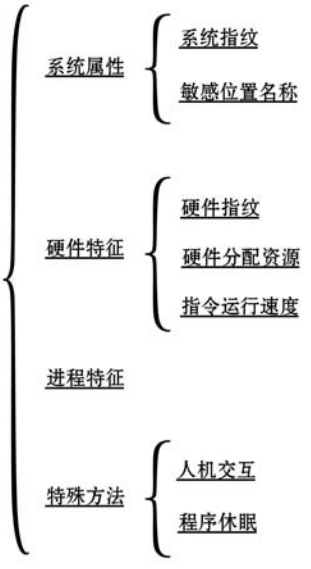

图 1　针对虚拟机检测方法的分类

根据虚拟机操作系统的属性进行探测是恶意代码检测虚拟机的各类方法中手段最为丰富的一种方法。在虚拟机操作系统内,注册表、文件系统、进程、服务中存在各种虚拟机指纹特征。据统计,安装在 Vmware 上的 WindowsXP 虚拟机有超过 50 个与“Vmware”、“vmx”有关的文件,在注册表中有超过 500 项键或键值与“Vmware”有关。此外,分析人员可能为用于恶意代码分析的虚拟机设定特别的用户名或重新命名恶意代码程序,因而恶意代码可以通过将这些敏感位置的名称与自定义的黑名单列表匹配,来检测是否为虚拟机。

虚拟机的硬件并非真实的硬件,因而通过模拟出的硬件信息也能够探测虚拟机。和操作系统一样,硬件也存在多种多样的指纹信息,其指纹检测包括对 CPU、BIOS、硬盘类型的判断以及对 CPU 名称、网卡地址、网卡名称、可插拔硬件设备的检测。除此之外,考虑到为虚拟机分配的硬件资源往往小于真实计算机的

硬件资源,因而可以通过对 CPU 核心数、内存大小、硬盘容量的检测来判断是否为虚拟机。在虚拟机中,指令运行的速度远远慢于真实系统,因而部分恶意代码通过计算指令运行的时间来检测虚拟机。

基于进程的检测是恶意代码对自身运行时虚拟内存空间的探测,常见的方法是检测中断描述符表(IDT)、全局描述符表(GDT)以及局部描述符表(LDT)在内存中的地址。真实系统中,这些描述符表的位置基本在确定的地址附近,且完全不同于虚拟机,因而具有较高的检测正确率。

最后,是对特殊检测方法的汇总。反图灵测试是主动探测人机交互的行为,在一定的时间间隔内,两次获取当前鼠标在屏幕上的位置坐标,比较两次位置是否发生变化,从而检测虚拟机。休眠是指在程序运行后,休眠一段较长的时间,再执行恶意行为,往往虚拟机在恶意代码真正运行前已停止工作,因而可以有效地躲避分析。

2 对抗沙箱检测方法

2.1 angr 介绍

angr 是美国圣巴巴拉大学 ShellPhish 团队研发的一款二进制分析框架,其源代码由 Python 编写,在 2015 年项目得到开源,目前项目已迭代至第七个版本。angr 可以实现动静态符号执行、控制流平坦化、自动化生成 ROP 链,应用于漏洞挖掘与软件破解,同时也是 CTF 比赛中逆向分析的常用工具之一[3,12-13]。使用者通过导入 angr 库并编写 Python 脚本实现对二进制代码的加载与分析。

angr 由以下几个子模块组成,每个子模块都可以独立加载:

CLE(CLE Loads Everything)将可执行程序加载进内存空间。例如,对于 EXE 程序,通过解析文件头,获得加载地址、各段对应的文件与内存的映射关系,进而将主文件加载至虚化的内存空间。接着,递归加载程序需要的 dll 文件。

Archinfo 针对不同架构的指令集,如 x86、ARM、MIPS、PowerPC 等,提供专有的类。每个类中都涉及寄存器类型、处理器位数、数据在内存中的存储方式以及是否支持 VEX、Capstone、Unicorn、KeyStone 等特性。

PyVEX 可以将机器代码转化为 Valgrind 使用的中间语言 VEX。通过 VEX 模拟执行程序的代码并修改内存、寄存器的值。

Claripy 是一个符号执行求解器,其后端为 Z3。

利用 angr 编写 Python 脚本,对二进制程序进行分析的整体流程如图 2 所示。首先,通过 Project 方法,加载程序至内存空间,并得到包含加载对象名称、各段地址空间、导入函数地址空间等属性的 project 对象。接着,利用 State 方法初始化程序堆栈以及寄存器,得到可以动态执行的对象 SimState。通过 SimState 对象可以查看、修改、符号化内存空间以及寄存器的值,并且提供用于保存用户自定义属性的插件。由于动态符号执行会产生分支,得到多个 Simstate 对象,仿真管理器(Simulation Managers)提供了对所有分支进行访问以及管理的功能。此外,仿真管理器在程序运行到某个地址或满足某种条件时,停止对应分支执行或者丢弃分支。

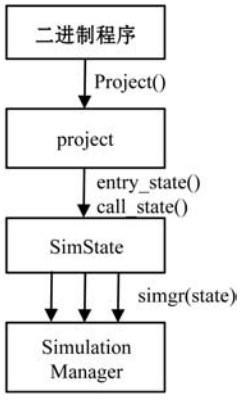

图 2 基于 angr 的程序分析流程

2.2 angr 对抗沙箱检测方法的原理

具有沙箱检测功能的恶意代码,大部分结合了十种以上的检测机制,基于符号执行的特点,会带来路径爆炸的问题,淹没恶意行为的分支。由于 angr 动态的模拟程序运行,可以方便地对 Win32 API 函数挂钩、重写 VEX 指令、完善内存结构。因而,针对于各种恶意代码对沙箱的探测,都可以采取措施进行对抗。

2.2.1 Win32 API 挂钩

恶意代码与普通 EXE 程序一样,在运行时动态链接系统提供的库函数。angr 在模拟程序动态运行时,提供了两种方法实现 Win32 API 函数。由于 CLE Loader 已经加载主程序以及其导入的 dll 文件至相应的地址空间,因而可以直接运行 API 函数对应的代码。对于大小写变换、字符串比较型的函数,比如 tolower()、wcsstr()、lstrcmpiA(),这种方法可以达到很好的效果。但是,对于大多数需要系统信息或者进行内核函数调用的函数,比如 CreateFileA()、GetModuleHandleA(),往往会产生众多分支,导致路径爆炸甚至出现分支异常终止。第二种方法是对 Win32 API 函数挂钩。首先,通过构建 SimProcedure 类,并在类中定义与实际 API 函数具有相同参数数量以及类型的 run 函数模拟

API 的实现。在程序加载后,对 project 对象使用 hook_symbol 方法替换 Win32 API 函数。在程序执行时,当指令指针指向 Win32 API 函数的首地址时,实际执行的是 run 函数中的 Python 代码。

注册表、文件、进程、服务的指纹特征是恶意代码检测沙箱最为常见的方法。表 1 展示了沙箱检测方法对应的 API 函数调用序列。挂钩这些函数并修改返回参数以及返回值即可绕过沙箱检测。

表 1　沙箱检测方法调用的 API 函数序列

沙箱检测方法	函数序列
注册表键	RegOpenKeyEx
注册表键值	RegOpenKeyEx -> RegQueryValueEx
文件/文件夹	[(GetLogicalDriveStrings/SHGetSpecialFolderPath() -> PathCombine] -> GetFileAttributes
进程 1	CreateToolhelp32Snapshot -> Process32First -> Process32Next
进程 2	GetProcessIdFromName
服务	OpenSCManager -> EnumServicesStatusEx

以 RegOpenKeyExA 函数为例,查阅微软 Win32 API 手册,其参数分别为主键(hKey)、子键(lpSubKey)、ulOptions、访问权限(samDesired)、注册表句柄(phkResult)。如图 3 所示。

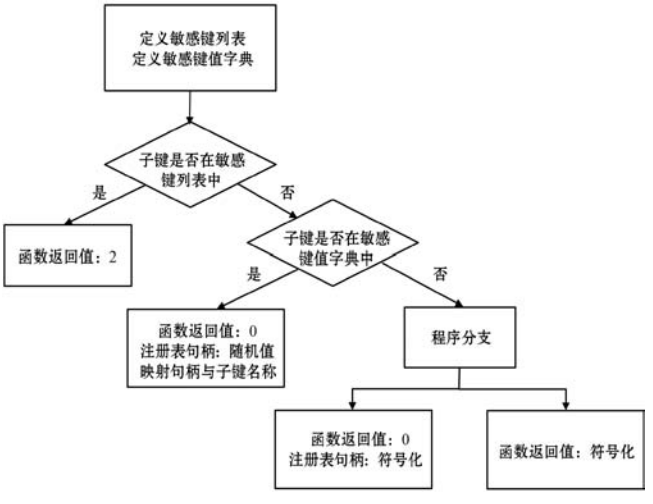

图 3　RegOpenKeyExA 函数的挂钩流程

函数存在以下三种情况:

(1) 首先定义敏感键列表,包括 vm、vmware、virtualbox、qemu 等对注册表检测的核心词语。接着,传入的参数子键与敏感键列表匹配,如果匹配成功,则函数返回值为 2(ERROR_FILE_NOT_FOUND),表示注册表项不存在。这种方式,可以成功对抗恶意代码对注册表特定键值的检测。如果匹配失败,则执行第二种情况。

(2) 首先定义敏感键值字典,字典中的键表示注册表的子键,值表示注册表的键名和键值,比如"HARDWARE\DEVICEMAP\SCSI\SCSI PORT 0\SCSI BUS 0\TARGET ID 0\LOGICAL UNIT ID 0': {'IDENTIFIER': 'INTEL'}"。接着,比较传入参数的子键是否等于敏感键值字典中的键,如果相等,则随机产生注册表句柄值并且函数返回值为 0,表示注册表存在并且成功打开。

(3) 如果前两种情况都不满足,根据符号执行的特点,将程序分支并符号化输出。对于 RegOpenKeyExA 函数,存在打开文件成功并写入注册表句柄以及打开文件未能成功两种情况。前一种情况,符号化句柄且函数返回值为 0;对于后一种情况,无需对句柄进行赋值,只需符号化返回值。

如表 1 所示,打开注册表键值需要两个函数,对于后序的 RegQueryValueEx 函数,其传入的注册表句柄参数是一个随机化的数值。为了获得其对应的子键名称,进而结合键名、键值两个参数与定义的敏感键值字典进行匹配,需要在 RegOpenKeyExA 函数中映射注册表句柄与子键名称。利用 angr 提供的自定义插件功能,可以为当前 SimState 增加属性,从而实现多个函数之间的状态共享。

2.2.2　VEX 指令修补

angr 在模拟程序动态执行时依据中间语言 VEX 完成各项操作。VEX 兼容大部分 x86 指令,实现对寄存器、内存的修改,但是对于某些特殊指令,如表所示,当程序分支执行这些指令时,该分支的状态变为错误并且停止运行。虽然 angr 对极少部分 VEX 指令,比如 CPUID 进行了修补,但不能满足实际恶意代码分析的需要,因而,我们对表 2 中的 VEX 指令都进行修补。

表 2　x86 特殊指令与 VEX 指令的对照表

x86 指令	VEX 指令	指令功能
CPUID	x86g_dirtyhelper_CPUID_sse2	获取 CPU 的信息
RDTSC	x86g_dirtyhelper_RDTSC	获取系统自启动以来运行的周期
SIDT	x86g_dirtyhelper_SxDT	获取中断描述符表的地址
SGDT	x86g_dirtyhelper_SxDT	获取全局描述符表的地址
SLDT	VEX 无法处理	

以指令 CPUID 为例,如图 4 所示。eax 寄存器是指令的输入参数,eax、ebx、ecx、edx 寄存器保存指令执行后返回信息,根据输入参数不同,可以获得 CPU 类型、制造商、商标、序列号、缓存等一系列信息。在沙箱检测中,当输入参数为 0x1 时,往往是对返回值中 ecx 寄存器的最高位:hypervisor CPU 位的检测,此时将 ecx

置为0,其他寄存器符号化。当输入参数为 0x40000000 时,执行指令获得 CPU 型号并保存在 ebx、ecx、edx 寄存器中。每个寄存器代表 4 个字符,共同构成 12 字符的型号名称。通过与"KVMKVMKVM"、"VMwareVMware"、"XenVMMXenVMM"等字符比较,进而判断是否为虚拟机环境。当输入参数分别为 0x80000002、0x80000003、0x80000004 且执行三次 CPUID 指令,可以获得共48 个字符的 CPU 名称。通过赋值普通字符,比如"Intel(R) Core(TM)2 CPU 6600 @2.4 GHz"可以对抗检测。

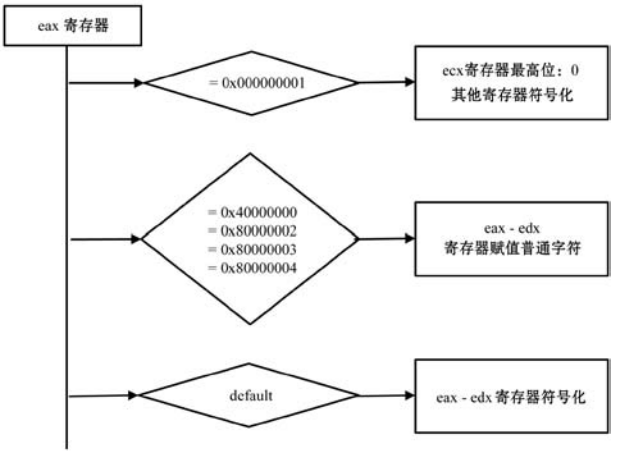

图 4 CPUID 指令执行流程

2.2.3 内存结构完善

angr 利用 state 方法初始化程序,设定寄存器以及内存空间的值,虽然可以实现代码执行,但不同于正常操作系统,没有在内存中设置完整的 PEB、TEB、TLS 等结构体。由于恶意代码通过读取结构体对应位置的值获取硬件信息,比如 PEB 进程环境块的 0x64 字节代表 CPU 数量,进而判断是否运行在虚拟机中。因而需要对 angr 的初始化文件打补丁,完善程序内存中的结构体。

2.2.4 angr 特性

angr 自身的特性同样能够对抗恶意代码对沙箱的部分检测方法。对于 dll 注入的检测,由于 angr 从外部实现对恶意代码执行流程的监控,其内存空间中不存在附加的 dll 函数。同时,angr 采用的 Win32 API 挂钩并没有改变内存空间中 API 函数的内容,从而绕过将 API 函数第一个操作码与跳转指令对应的操作码比较的检测方法。

通过上文所述的各种方法,可以对抗大多数恶意代码对沙箱的检测。对于未知的检测手段,基于 angr 动态符号执行的特点,需要做到在其所有分支路径中包含恶意行为的分支并且尽可能减少产生的分支数目。考虑到执行 Win32 API 函数内部的代码会造成路径爆炸,因而挂钩所有 Win32 API 函数(除 2.2.1 节描述的字符串变换、比较型的函数),且在函数体内不执行任何操作,仅仅将函数的返回值设定为符号值。并且由于 angr 将未赋值的内存空间作为符号值代入动态执行,虽然可能增加路径分支的数目,却可以确保恶意行为分支不被遗漏。

2.3 angr 对抗沙箱检测方法的实现

基于 angr 对抗恶意代码沙箱检测的原型系统的实现如图 5 所示。首先,在程序加载时,将 auto_load_libs 以及 except_missing_libs 两个参数设置为真,并指定加载 dll 文件的路径;其次,利用 analyses 对象中提供的 CalleeCleanupFinder 方法,对所有导入的 API 函数挂钩,挂钩的函数体内不执行任何操作;接着,将 Win32_API_Hooking.py 文件中定义的函数挂钩程序,对于同一个 API 函数,这一步的挂钩将覆盖 CalleeCleanupFinder 中的挂钩;之后,利用 angr 提供的 SimStatePlugin 对象,注册可以增加 SimState 属性的插件。最后,使用动态符号执行模拟程序的运行。

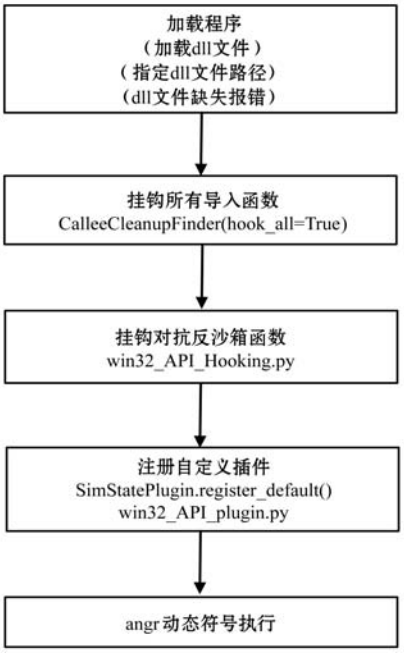

图 5 对抗恶意代码沙箱检测的原型系统

对于 2.2 节中提到的方法,在原型系统实现时,对于 Win32 API 函数挂钩,只需要在 Win32_API_Hooking.py 文件中创建与 API 函数名称一致的类并实现函数执行的方法,就可以通过 angr 提供的接口函数 hook_symbol 将 API 函数挂钩。而对于 VEX 指令修补以及内存结构完善两种方法,需要通过修改 angr 的源代码,从而实现功能,对应修改的文件分别是 angr/engines/vex/dirty.py 以及 angr/simos/windows.py。

表 3 列举了原型系统对抗沙箱检测各类方法时采取的应对措施。考虑到不同指令、函数功能上的差异,相应的修补方式也不尽相同,修补代码的实现遵循这两条原则:对于单指令、单函数的修补,首先判断输入

的参数与沙箱检测时使用的参数是否一致,如果能够匹配,使用对抗沙箱检测的方法,否则符号化函数的输出参数以及返回值;对于序列化的函数,采用的修补方式不仅需要满足单函数修补原则,并且利用插件功能记录程序执行的状态,后序函数实现时需要考虑前序函数及其参数是否已经满足沙箱检测的条件。

表 3　对抗沙箱检测的方法

沙箱检测	对抗方法	具体实现
指令运行时间 1	VEX 指令修补	RDTSC 指令:系统时间每次增加 500
CPU hypervisor 位	VEX 指令修补	CPUID 指令:ecx 寄存器最高位为 0
CPU 商标	VEX 指令修补	CPUID 指令:执行 3 次,每次通用寄存器组成 16 个字符
IDT/GDT	VEX 指令修补	设定 SIDT/SGDT 指令得到的地址
CPU 数量 1	内存结构完善	PEB 第 0x64 字节值大于等于 2
TLS	内存结构完善	fs 寄存器的第 0x44 字节
CPU 数量 2	API 挂钩	GetSystemInfo
硬盘空间 1	API 挂钩	CreateFile -> DeviceIOControl
硬盘空间 2	API 挂钩	GetDiskFreeSpaceExA
硬盘驱动	API 挂钩	SetupDiGetClassDevs -> SetupDiEnumDeviceInfo -> SetupDiGetDeviceRegistryProperty
内存空间	API 挂钩	GlobalMemoryStatusEx
硬盘属性	API 挂钩	IsNativeVhdBoot
SMBIOS 属性	API 挂钩	GetSystemFirmwareTable
ACPI 属性	API 挂钩	EnumSystemFirmwareTables -> GetSystemFirmwareTable
电源属性	API 挂钩	GetPwrCapabilities
网卡地址/名称	API 挂钩	GetAdaptersAddresses/GetAdaptersInfo
调试器 1	API 挂钩	IsDebuggerPresent
调试器 2	API 挂钩	SetLastError -> OutputDebugString -> GetLastError
人机交互(鼠标位置)	API 挂钩	GetCursorPos
系统用户名	API 挂钩	GetUserName
程序运行路径	API 挂钩	GetModuleFileName
共享文件夹	API 挂钩	WNetGetProviderNameA
窗口	API 挂钩	FindWindow
休眠检测/系统运行时间	API 挂钩	GetTickCount -> sleep
程序等待时间 1	API 挂钩	CreateEvent -> WaitForSingleObject
程序等待时间 2	API 挂钩	IcmpCreateFile -> IcmpSendEcho

续表 3

沙箱检测	对抗方法	具体实现
WMI	API 挂钩	CoInitializeEx -> CoInitializeSecurity -> CoCreateInstance -> IWbemLocator. ConnectServer -> CoSetProxyBlanket -> IWbemServices. ExecQuery -> IEnumWbemClassObject. Next -> IWbemClassObject. Get
注:此表省略了表 1 中的内容		

3　测试与分析

基于 angr 对抗沙箱检测方法的原型系统使用最新的 angr7,并且通过 OSX Sierra 系统下 Python2.7 中的测试。用于 angr 加载恶意代码的 dll 文件拷贝于 32 位 Windows XP SP3 系统中的 system32 文件夹。al-khaser 以及 paranoid fish 是两款用于对沙箱、虚拟机进行检测的开源项目,均使用 C ++ 编程实现,其搜集并总结了 122 类恶意代码使用的方法,并且 al-khaser 仍处于不断更新的状态。为了便于分析,我们使用 32 位编译器生成用于分析的 EXE 文件。

微步科技提供的微步云沙箱,内核集成了人机交互、反沙箱检测、内核级分析、网络模拟等多个高级分析模块。CrowdStrike 公司的 VxStream 沙箱是基于机器学习的下一代恶意软件扫描器。值得一提的是,可以在其扫描配置中将反沙箱检测设为强等级。对于 al-khaser 以及 paranoid fish,我们分别使用布谷鸟沙箱 2.0.4 版本、微步云沙箱以及强反沙箱检测等级的 VxStream 进行扫描,其分析环境均为 Windows 7,与原型系统分析比较的结果如表 4 所示,其中 T 代表恶意代码检测出沙箱环境、F 代表未能检测出沙箱环境。为了简化表格,表 4 中没有列出四款系统均能识别的沙箱检测技巧以及 paranoid fish 与 al-khaser 中重复的检测技巧。

表 4　针对 al-khaser 以及 paranoid fish 的检测结果

检测函数名称	布谷鸟	微步	VxStream	原型系统
al -> virtual_pc_reg_keys	T	F	F	F
al -> vbox_reg_keys	T	F	F	F
al -> vbox_files	T	F	F	F
al -> vbox_check_mac	T	T	F	F
al -> vbox_devices	T	F	F	F
al -> vbox_processes	T	F	F	F
al -> vbox_devices_wmi	T	T	F	F
al -> vbox_mac_wmi	T	T	F	F

续表 4

检测函数名称	布谷鸟	微步	VxStream	原型系统
al -> vbox_eventlogfile_wmi	T	F	F	F
al -> vbox_firmware_SMBIOS	T	T	F	F
al -> vbox_firmware_ACPI	T	T	T	F
al -> NumberOfProcessors	T	T	F	F
al -> ldt_trick	F	F	F	T
al -> number_cores_wmi	T	T	T	F
al -> dizk_size_deviceiocontrol	F	T	F	F
al -> setupdi_diskdrive	T	T	F	F
al -> disk_size_getdiskfreespace	F	T	F	F
al -> cpuid_is_hypervisor	T	T	F	F
al -> cpuid_hypervisor_vendor	T	T	T	F
al -> serial_number_bios_wmi	T	T	F	F
al -> model_computer_system_wmi	T	T	F	F
al -> manufacturer_computer_system_wmi	T	T	F	F
al -> current_temperature_acpi_wmi	T	T	T	F
al -> GetPwrCapabilities	T	T	T	F
pa -> cpu_rdtsc	T	T	T	F
pa -> cpu_rdtsc_force_vmexit	T	T	T	F
pa -> check_hook_ShellExecute_m1	T	F	F	F
pa -> vbox_network_share	T	F	F	F
检出率	25/122	19/122	7/122	1/122

从表 4 的结果可知,相比于开源的布谷鸟平台,微步云沙箱以及 VxStream 沙箱在对抗恶意代码检测沙箱环境方面均有一定的提升,特别是设置了强反沙箱等级的 VxStream 检出率仅仅为 7/122。但是对于由汇编代码直接获取系统信息的检测方法,比如 cpuid、rdtsc,这几种沙箱都未能成功对抗。本文设计的原型系统能够成功绕过大部分检测方法,因而显著优于现有的在线沙箱检测平台。

4 结 语

本文基于 angr 提出了一种能够对抗恶意代码针对沙箱环境进行探测的二进制代码分析方法。原型系统采用 Win32 API 函数挂钩、VEX 指令修补以及内存结构完善三种方法,针对常见的恶意代码反沙箱机制采取相应的修补措施。实验表明,本文提出的方法在对抗恶意代码沙箱检测方面,优于其他传统的方法。因此,在本文的基础上,利用 angr 动态符号执行的特点,自动化地捕获恶意代码的行为特征是我们下一步的研究工作。

参 考 文 献

[1] McAfee. McAfee labs threats report: June 2017[EB/OL]. (2017-07)[2018-06-10]. https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf.

[2] Kruegel C. Labs report at RSA: Evasive malware's gone mainstream[EB/OL]. (2015-04)[2018-06-10]. https://www.lastline.com/labsblog/labs-report-at-rsa-evasive-malwares-gone-mainstream.

[3] Shoshitaishvili Y, Ruoyu W, Christopher S, et al. (State of) The art of war: offensive techniques in binary analysis[C]// Proceedings of the IEEE Symposium on Security and Privacy (S&P). SAN JOSE, California, USA, 2016: 138-157.

[4] Brumley D, Hartwig C, Liang Z, et al. Automatically identifying trigger-based behavior in malware[M]//Botnet Detection. Springer US, 2008: 65-88.

[5] Baldoni R, Emilio C, Daniele C D, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys, 2018, 51(3): 50-89.

[6] Yan L, Manjukumar J, Mu Z, et al. V2E: combining hardware virtualization and softwareemulation for transparent and extensible malware analysis[J]. ACM Sigplan Notices, 2012, 47(7): 227-238.

[7] 于航, 刘丽敏, 高能, 等. 基于模拟器的沙箱系统研究[J]. 信息网络安全, 2015(9): 139-143.

[8] Alexander C, Stanislav S. Defeating sandbox evasion: how to increase successful emulation rate in your virtualized environment[C]//Virus Bulletin International Conference. Denver, USA, 2016: 1-5.

[9] Martignoni L, Roberto P, Giampaolo F R, et al. Testing cpu emulators[C]//Proceedings of the International Symposium on Software Testing and Analysis(ISSTA). Chicago, USA, 2009: 261-272.

[10] Raffetseder T, Christopher K, Engin K. Detecting system emulators[C]//the Information Security Conference(ISC). Valparaiso, CI, 2007: 1-18.

[11] Ferrie P. Attacks on more virtual machine emulators[J]. Symantec Technology Exchange, 2007, 55: 1-17.

[12] Bao T, Ruoyu W, Yan S, et al. Your exploit is mine: automatic shellcode transplant for remote exploits[C]//Proceedings of the IEEE Symposium on Security and Privacy(S&P). SAN JOSE, California, USA, 2017: 824-839.

[13] Shoshitaishvili Y, Ruoyu W, Christophe H, et al. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]//Proceedings of the Network and Distributed System Security Symposium(NDSS). San Diego, California, USA, 2015: 1-15.