VMDE

Virtual Machines

Detection Enhanced

N. Rin
EP_X0FF

November 2013

Abstract

This document contains short overview of existing and exploited by WinNT malicious software (*malware*) methods (*AntiVM*) that help malware detect execution in the controlled environment such as virtual machine (*VM*) or/and sandbox. However, this is not complete R&D of each malware with AntiVM, we will focus only on most popular and often seen methods. Unlike other articles, we will show other methods that detect presence of the software VM. We assume readers are familiar with Windows NT based malware, Windows NT core components, x86 architecture and programming languages.

VM detection code criteria:

- User mode execution only;
- No requirement for elevation of privileges;
- Detection must work with all popular virtualization products.

Target software (only popular malware targeted WinNT software):

- VMware Workstation;
- Parallels Desktop;
- Oracle VirtualBox;
- Sandboxie;
- Microsoft Virtual PC (former Connectix Virtual PC), also known as Windows XP Mode in the Windows 7 release.

1. Overview

Virtual machines detection based mostly on execution artifacts with predicted behavior. They can be software-based and hardware-based. Typical example of hardware-based detection is *timing* attack. This is a hardware specific method based on prediction that code inside virtual machine will execute slower than on real. As example: we calculate time between start and end of code block execution using ***rdtsc*** instruction. If the returned value exceeds a predetermined barrier then this used as strong indication of VM presence.

Next malware authors are trying to incorporate hardware-based detections, most of them developed years ago as proof-of concepts. All methods here based on execution of specific instructions:

- ***sldt***
- ***sgdt***
- ***sidt*** [1]
- ***str*** [2]
- ***smsw*** [3]

After execution, malware performs analysis of the returned data. As for now, virtualization software made a big step forward and these methods simple do not work at all. Malware looks for a specific user names, loaded libraries, registry keys and values, process names, Windows product IDs, filenames. Let us take a brief overview of most notorious malware families with AntiVM. An example of attempt to use hardware-based detection is **Win32/Conficker**, which abuses ***sldt*** instruction trick to detect VM. In continuation, we have **Win32/Simda**, which contains special VM detection mechanisms: *Simda* uses blacklist of Product IDs to ban most popular online sandboxing services like Anubis or JoeBox, scans Windows registry for specific installed software and check if the several VM service processes are running. **Win32/Zeus** clone known as *Citadel* brings special signature scanning to detect running on the virtual environment and depending on that fact change it execution flow. It implemented as scanning of image resources for specific string patterns such as "VMware", "virtualbox" and others. One of **WinNT/Alureon** (family also known as *TDSS/TDLX*) members, a fork of *TDL4* project *MaxSS* implements VM detection by using documented WMI queries[4]. It selects information from various WMI providers such as Win32_Processor, Win32_SCSIController, Win32_ComputerSystem, Win32_DiskDrive, Win32_BIOS, and then compares data with it blacklist. Yet another example of well-known malware family with VM detection is **Win32/Sirefef**. Some of it droppers were using specific VM detection code based on documented VM features (querying VMware hypervisor port), and not documented but known VM behavior (Virtual PC way of communication with host). Number of ransomware trojans, as for example **Win32/CBeplay** family is widely using VM detection tricks to complicate analysis. Some of malware may abuse different fact of virtual machines, such as limited virtual disk size,

[1] http://web.archive.org/web/20070911024318/http://invisiblethings.org/papers/redpill.html

[2] http://charette.no-ip.com:81/programming/2009-12-30_Virtualization/www.s21sec.com_VMware-eng.pdf

[3] http://www.offensivecomputing.net/dc14/vmdetect.cpp

[4] http://social.technet.microsoft.com/wiki/contents/articles/942.hyper-v-how-to-detect-if-a-computer-is-a-vm-using-script.aspx

known Ethernet adapters MAC addresses, hard drive/videocard vendor names, and logical bombs.

To detect execution in the sandbox (e.g. Sandboxie) malware may perform self-checking for API splicing. Another good self-explaining example found inside leaked **Win32/Carberp** trojan source code, shown on Figure 1-1.

```
bool DetectVM()
{
    #define BUF_SIZE 1024
    static char szBuf[BUF_SIZE] = {'\0'};
    if (!GetSystemDir(szBuf, MAX_PATH)) return true;

    if (CheckDir(szBuf, "prleth.sys")) return true;
    if (CheckDir(szBuf, "hgfs.sys")) return true;
    if (CheckDir(szBuf, "vmhgfs.sys")) return true;

    if (GetModuleHandle("dbghelp.dll")) return true;
    if (GetModuleHandle("sbiedll.dll")) return true;

    DWORD dwUsl = BUF_SIZE;
    if (!GetUserName(szBuf, &dwUsl)) return true;
    if (0 == lstrcmp(szBuf, "CurrentUser")) return true;
    if (0 == lstrcmp(szBuf, "Sandbox")) return true;

    dwUsl = BUF_SIZE;
    if (!GetComputerName(szBuf, &dwUsl)) return true;
    if (0 == lstrcmp(szBuf, "SANDBOX")) return true;

    if (!CheckReg("HARDWARE\\DESCRIPTION\\System",
                  "SystemBiosVersion", szBuf, BUF_SIZE)) return true;
    if (STR::Pos("VBOX", szBuf)) return true;

    if (!CheckReg("HARDWARE\\DESCRIPTION\\System",
                  "VideoBiosVersion", szBuf, BUF_SIZE)) return true;
    if (STR::Pos(szBuf, "VirtualBox") >= 0) return true;

    if (!CheckReg("SOFTWARE\\Microsoft\\Windows\\CurrentVersion",
                  "SystemBiosVersion", szBuf, BUF_SIZE)) return true;
    if (STR::Pos(szBuf, "55274-640-2673064-23950") >= 0) return true;
    if (STR::Pos(szBuf, "76487-644-3177037-23510") >= 0) return true;
    if (STR::Pos(szBuf, "76487-337-8429955-22614") >= 0) return true;

//  if (IsVPC()) return true;
    //if (IsVMWare()) return true;

    return false;
}
```

FIGURE 1-1. Fragment of Win32/Carberp code.

Win32/Caphaw implements detection by using files and processes scanning. On Figure 1-2 shown Delphi module, code from it widely used in low cost malware such as ransomware **Win32/Kovter** and various IRC bots (original author comments unmodified).

```
function CheckAnti: Boolean;
var
  Path: string;
begin
  result := false;
  Path := ExtractFilePath(ParamStr(0));
  if (processexists('joeboxcontrol.exe')) //JoeBox
    or (processexists('joeboxserver.exe')) //Joebox 2
    or (processexists('wireshark.exe')) // WireShark
    or (processexists('regmon.exe')) //Regmon
    or (processexists('filemon.exe')) //FileMon
    or (processexists('procmon.exe')) //ProcMon
    or (processexists('VBoxService.exe')) //Vbox

  or (modulecheck('SbieDll.dll')) //Sandboxie
    or (modulecheck('api_log.dll')) //SunBelt
    or (modulecheck('dir_watch.dll')) //Sulbelt's Sandbox

  or (IsUsername('username')) //ThreadExpert
    or (IsUsername('USER')) //Sandbox
    or (IsUsername('user')) //Sandbox 2
    or (IsUsername('currentuser')) //Normal

  or (Pos('c:\insidetm', Path) <> 0) //Anubis
    or (DirEctoryExists('C:\analysis')) // Sunbelt 3
    or (DeBuggerPresent = true) //Debuggers
    or (InVmWare = True) //VmWare
    or (IsInVPC = True)
    then
    result := true
end;
```

FIGURE 1-2. Code fragment used in low cost malware.

Win32/Gamarue also known as *Andromeda* performs several checks to determine execution environment. It employs Sandboxie detection using almost documented way - checking presence of loaded SBIEDLL.DLL[5] and detects VM by comparing disk ID strings with predefined product blacklist. Example of usual Sandbox detection seen in malware shown on Figure 1-3 (original author comments unmodified).

[5] http://www.sandboxie.com/index.php?SBIE_DLL_API

```
BOOL IsInSandbox(){
    HKEY    hOpen;
    char    *szBuff
    int     iBuffSize;
    HANDLE  hMod;
    BOOL    bResult= FALSE;
    LONG    nRes;

    szBuff= (char*)calloc(512, sizeof(char));

    hMod= GetModuleHandle("SbieDll.dll");  //Sandboxie
    if( hMod!=0 ){
        bResult= TRUE;
        return bResult;
    }
    hMod= GetModuleHandle("dbghelp.dll"); // Thread Expert
    if( hMod!=0 ){
        bResult= TRUE;
        return bResult;
    }
    nRes= RegOpenKeyEx(HKEY_LOCAL_MACHINE, "Software\\Microsoft\\Windows\\CurrentVersion", 0L, KEY_QUERY_VALUE, &hOpen);
    if( nRes==ERROR_SUCCESS ){
        iBuffSize= SizeOf(szBuff);
        nRes= RegQueryValueEx(hOpen, "ProductId", NULL, NULL, (unsigned char*)szBuff, &iBuffsize);
        if( nRes==ERROR_SUCCESS ){
            if( strcmp( szBuff, "55274-640-2673064-23950" )==0 ){ //Joe Box
                bResult= TRUE;
            }else if( strcmp( szBuff, "76487-644-3177037-23510" )==0 ){ //CW Sandbox
                bResult= TRUE;
            }else if( strcmp( szBuff, "76487-337-8429955-22614" )==0 ){ //Anubis
                bResult=TRUE;
            }else{
                bResult= FALSE;
            }
        }
        RegCloseKey(hOpen);
    }
    return bResult;
}
```

FUGURE 1-3. Code fragment with usual Sandboxie detection and online sandbox detection using Windows ProductId.

Win32/Winwebsec is infamous Fake AV that implements detection by using ***cpuid*** instruction to get hypervisor vendor name and then compare it with blacklisted VMware vendor string[6]. Additionally some variants of Winwebsec may use Setup API[7] to query specific hardware information, code fragment shown Figure 1-4 (original author comments unmodified). Malware detecting usual set of popular VM and QEMU emulator because this software used by several online sandbox services such as Anubis.

[6] http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1009458

[7] http://msdn.microsoft.com/en-us/library/cc185682(v=vs.85).aspx

```
bool VmDetect()
{
    //char ids[]="VBOX HARDDRIVE,QEMU HARDDISK,VMWARE VIRTUAL IDE HARD DRIVE,VIRTUAL HD";
    char vbox[]="VBOX";
    char qemu[]="QEMU";
    char vmw[]="VMWARE";
    char unkn[]="VIRTUAL HD";
    GUID h=GUID_DEVCLASS_DISKDRIVE;
    DWORD rg,rsz;
    char *buf=(char*)malloc(4096);
    SP_DEVINFO_DATA devinfo;
    devinfo.cbSize=sizeof(SP_DEVINFO_DATA);
    HDEVINFO devs=SetupDiGetClassDevs(&h,NULL,NULL,DIGCF_PRESENT);
    SetupDiEnumDeviceInfo(devs,0,&devinfo);
    SetupDiGetDeviceRegistryPropertyA(devs,&devinfo,SPDRP_FRIENDLYNAME,&rg,(BYTE*)buf,4096,&rsz);
    //MessageBoxA(0,buf,"",MB_OK);
    if(strstr(buf,vbox)!=NULL ||
        strstr(buf,qemu)!=NULL ||
        strstr(buf,vmw)!=NULL || strstr(buf,unkn)!=NULL)
    {
        free(buf);
        return true;
    }
    else
    {
        free(buf);
        return false;
    }
}
```

FIGURE 1-4. Win32/Winwebsec VM detection routine.

All the above methods widely used in various malware families and malware obfuscation software (**VirTool/Obfuscator**). Summarized VM / Sandbox detection methods listed in Table 1-1.

TABLE 1-1. Summary of most popular VM / Sandbox detection methods.

Method	VM	Sandbox
CPU instructions (sidt, sgdt, sldt, str, smsw, cpuid, rdtsc, in)	+	-
User/Computer names	-	+
Running processes	+	+
Running services	+	+
Loaded modules and memory scanning	+	+
Loaded drivers and their devices	+	+
Registry keys and their values	+	+
Files/Directories	+	+
Named system objects (Devices, Drivers, Mutants, Semaphores, Events, Sections, Ports)	+	+
MS Hydra GUI subsystem (window names, class names)	+	+
IOCTL requests	+	-
Hardware information (Disk serial numbers, HDD size, HWID's, vendor names, Ethernet MAC addresses)	+	-
Windows Product ID's	+	-
WMI queries	+	+
Own code modification detection (API hooking detection)	-	+
Logical bombs (user desktop contents, mouse movements)	-	+

Conclusion: many malware currently has VM detection on board in various implementations of the same methods. This slowdowns analysis and sample may behave completely different inside virtual machine if it detects it.

2. Detection

Now let us take a look how we can implement comprehensive VM detection. The question here "Is it possible at all"? Short answer – no. However, based on our knowledge and set of software we must detect we will try to implement generic methods for VM detection. However, they will not be heuristic, as we cannot rely on software implementation bugs and matching our criteria from Abstract. It is always better abuse things that cannot be simple fixed without product code revision.

Which possible VM detection vectors we decided to use:

- Software environment (2.1);
- Emulated hardware (2.2);
- Additional execution artifacts (2.3).

VM environment usually set up by installing inside VM guess-to-host virtualization support software. It is VMware Tools, VirtualBox Guess Additions, and Virtual PC VM Additions. Each of them contains kernel mode drivers and this fact gives us a good point to start.

What we decided to throw away:

A. Process names, driver names, file names, known registry values and names, computer or user names;

B. VM detection proof-of-concepts;

C. GUI based detections;

D. Hardware access related detections (MAC addresses, HDD related info);

E. WMI;

F. Logical bombs.

Why:

A. The above methods are unstable and depend on initial VM configuration. If user decided to not use VM additions all these methods will fail;

B. They do not work with modern hardware supported virtualization, do not work at all or too specific;

C. Is too specific and almost the same as A case;

D. Noticeable and unreliable;

E. Significant code increase only to perform basic queries;

F. In most cases, they are unreliable.

We created special tool called VMDE designed to fulfill our criteria and next we will take a closer look on it with examples from VMDE code.

2.1. Software environment

2.1.1. Documented and semi-official detection ways

VMware provides official documented ways of VM detection. First way is based on using hypervisor port **0x5658** ("**VX**") and hypervisor magic DWORD **0x564D5868** which stands for "**VMXh**". This method is very old known and widely used by malware. Figure 2-1 shows method implementation inside our VMDE.

```
    if ( IsVM == 0 ) {
        /*
           query vmware presence by hypervisor port
           http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1009458
        */
        __try {
            __asm push   edx
            __asm push   ecx
            __asm push   ebx
            __asm mov    eax, 'VMXh'
            __asm mov    ebx, 0
            __asm mov    ecx, 10
            __asm mov    edx, 'VX'
            __asm in     eax, dx
            __asm cmp    ebx, 'VMXh'
            __asm setz   [IsVM]
            __asm pop    ebx
            __asm pop    ecx
            __asm pop    edx
        }
        __except(EXCEPTION_EXECUTE_HANDLER)
        {
            IsVM = 0;
        }
    }
```

FIGURE 2-1. Official VMware detection by using hypervisor port and magic DWORD.

The above detection method (as well as used in ScoopyNG[8]) can be simple turned off by VM reconfiguration through %vmname%.vmx file[9] or by additional filtering[10]. However, this backdoor usually required for normal work of VMware Tools.

Second documented method described by both VMware and Microsoft. It is testing the CPUID hypervisor (HV) present bit. Quote from official VMware document:

Intel and AMD CPUs have reserved bit 31 of ECX of CPUID leaf 0x1 as the hypervisor present bit. This bit allows hypervisors to indicate their presence to the guest operating system. Hypervisors set this bit and physical CPUs (all existing and future CPUs) set this bit to zero. Guest operating systems can test bit 31 to detect if they are running inside a virtual machine.

Quote from official Microsoft documentation[11]:

Before it uses any hypervisor interface functions, software should first determine whether it runs within a virtualized environment. On x64 platforms, software verifies that it runs within a virtualized environment by executing the CPUID

[8] http://www.trapkit.de/research/vmm/scoopyng/

[9] http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf

[10] http://www.securitylab.ru/analytics/427855.php

[11] http://msdn.microsoft.com/en-us/library/windows/hardware/ff538624(v=vs.85).aspx

instruction with an input (EAX register) value of 1. When the CPUID instruction is executed, code should check bit 31 of register ECX. Bit 31 is the *hypervisor-present* bit. If the hypervisor-present bit is set, the hypervisor is present. In a non-virtualized environment, the hypervisor-present bit is clear.

Example of this detection shown on Figure 2-2.

```
BOOL IsHypervisor(
    VOID
    )
{
    int CPUInfo[4] = {-1};

    /*
       query hypervisor presence
       http://msdn.microsoft.com/en-us/library/windows/hardware/ff538624(v=vs.85).aspx
       be aware this detection can be bogus
    */

    __cpuid(CPUInfo, 1);
    if ((CPUInfo[2] >> 31) & 1) {
        return TRUE;
    }

    return FALSE;
}
```

FIGURE 2-2. HV detection by ECX bit.

If HV presence confirmed then it is good to know which type of HV we have. Quote:

Intel and AMD have also reserved CPUID leaves 0x40000000 - 0x400000FF for software use. Hypervisors can use these leaves to provide an interface to pass information from the hypervisor to the guest operating system running inside a virtual machine. The hypervisor bit indicates the presence of a hypervisor and that it is safe to test these additional software leaves. VMware defines the 0x40000000 leaf as the hypervisor CPUID information leaf. Code running on a VMware hypervisor can test the CPUID information leaf for the hypervisor signature. VMware stores the string "VMwareVMware" in EBX, ECX, EDX of CPUID leaf 0x40000000.

The above works for different HV as well.

```
BYTE GetHypervisorType(
    VOID
    )
{
    int CPUInfo[4] = {-1};
    char HvProductName[0x40];

    __cpuid(CPUInfo, 0x40000000);
    RtlSecureZeroMemory(HvProductName, sizeof(HvProductName));
    memcpy(HvProductName, CPUInfo + 1, 12);

    /* http://msdn.microsoft.com/en-us/library/windows/hardware/ff542428(v=vs.85).aspx */
    if (_strcmpiA(HvProductName, "Microsoft Hv") == 0) {
        return 1;
    }

    /* http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1009458 */
    if (_strcmpiA(HvProductName, "VMwareVMware") == 0) {
        return 2;
    }
    /* Parallels VMM ids */
    if (_strcmpiA(HvProductName, "prl hyperv") == 0) {
        return 3;
    }
    return 0;
}
```

FUGURE 2-3. Detecting type of HV.

The above ***cpuid*** method used in Win32/Winwebsec malware.

Microsoft Virtual PC formerly known as Connectix Virtual PC can be detected by abusing method it uses to communicate in it guest-to-host model. It is not documented anywhere, but so widely used and so old known, so we can say it is "semi-documented" way[12]. Virtual PC uses a bunch of invalid instructions to allow the interfacing between the virtual machine and the Virtual PC software. The detection code is very popular in ITW malware and malware authors usually blindly copy-past it even without changing set of invalid instructions. Usual set is the following: 0x0F 0x3F 0x07 0x0B. However exists more than 1000 combinations of 0x0F 0x3F 0xXX 0xXX that can be used to detect Virtual PC.

```
/*
   use well-known trick with illegal instructions, but be creative and don't use the
   same as here, there are numbers of them actually (> 1000), not only one set
*/
if ( IsVM == 0 ) {
    __try {
        __asm push   ebx
        __asm mov    ebx, 0
        __asm mov    eax, 1
        __asm __emit 0Fh
        __asm __emit 3Fh
        __asm __emit 0Dh
        __asm __emit 0h
        __asm test   ebx, ebx
        __asm setz   [IsVM]
        __asm pop    ebx
    }
    __except(VPCExceptionHandler(GetExceptionInformation())) {  }
}
```

FIGURE 2-4. Virtual PC detection by illegal instruction.

[12] http://www.codeproject.com/Articles/9823/Detect-if-your-program-is-running-inside-a-Virtual

2.1.2. Abusing named system objects

All software selected by our criteria has a bunch of named system objects. If user decides not install guess-to-host support VM software (Tools, Additions) the following detection will fail. However, usually these slowdowns VM work as emulated virtual devices work without proper drivers. Named system objects are perfect targets for the VM detection. Why objects are so important? Because Windows operation system object-based[13]. Drivers, devices, communication channels, sections, directories, symbolic links, synchronization mechanisms, jobs, etc, they all represented as objects in kernel mode. The Windows Object Manager controls objects that are part of the kernel-mode operating system. Objects are stored in the corresponding objects directory[14], for example, *KnownDlls* keeps section type objects of the system dll's and *Device* directory keeps device type objects created by system or third party drivers. Figure 2-5 shows typical object directory.

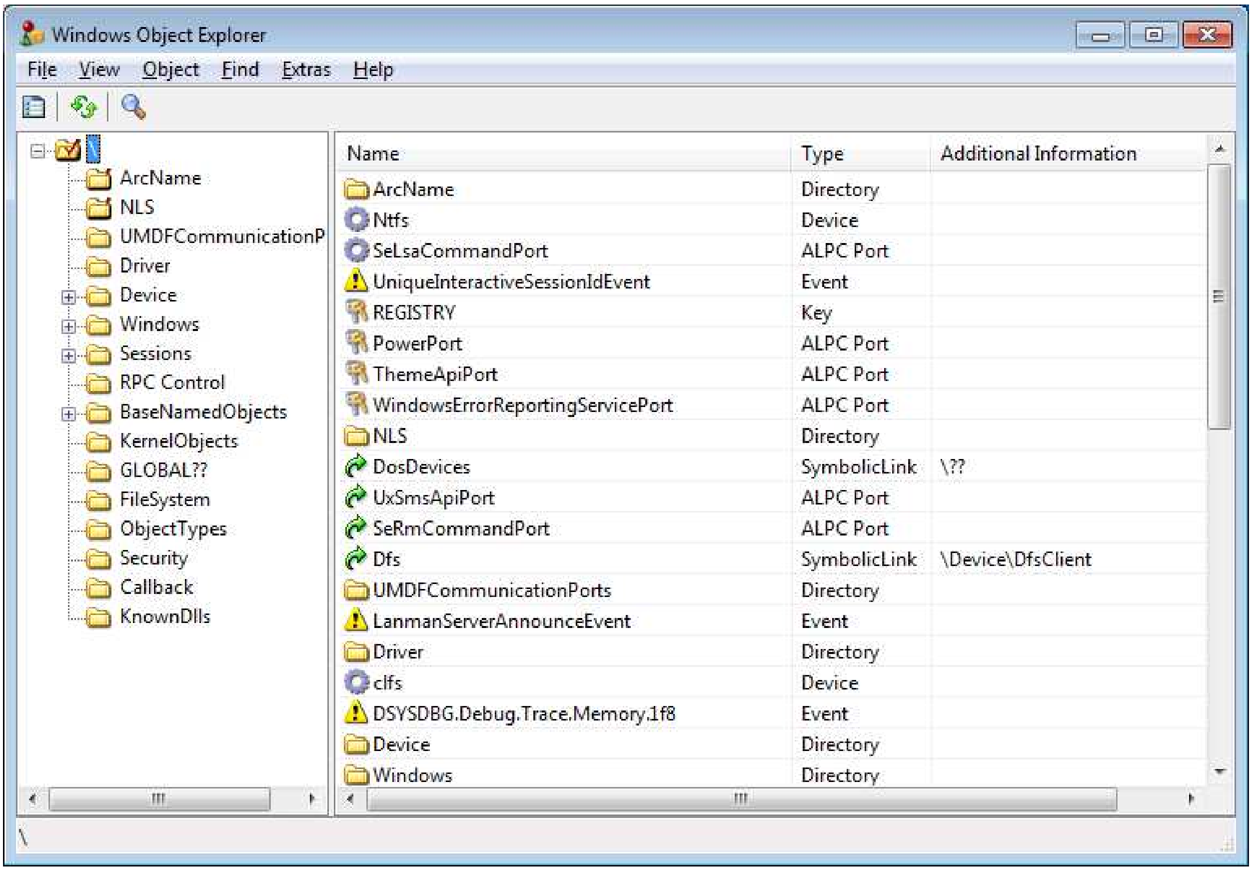

FIGURE 2-5. Objects directory root, Windows 7.

[13] http://msdn.microsoft.com/en-us/library/windows/hardware/ff554383(v=vs.85).aspx

[14] http://msdn.microsoft.com/en-us/library/windows/hardware/ff557755(v=vs.85).aspx

In a particular, we are interested in the following object types:

- Device names
- Mutants (Win32 API alias Mutex)
- Communication Ports

How malware usually use this? Mostly by calling Win32 API services like *CreateFile* or *CreateMutex*. Let us look how we can do the same on native level. As example, we will take detection of Virtual PC Windows XP Mode. Despite the fact that it is almost the same Virtual PC 2007, it has some new features that may be used to detect it exactly as "XP Mode". We only need to check if the mutex

MicrosoftVirtualPC7UserServiceMakeSureWe'reTheOnlyOneMutex

exists.

```
BOOL IsMutexExist(
    PWSTR MutexName
    )
{
    OBJECT_ATTRIBUTES attr;
    UNICODE_STRING ustrName;
    NTSTATUS Status;
    HANDLE hObject = NULL;

    WCHAR szBuffer[MAX_PATH] = {0};

    if (!ARGUMENT_PRESENT(MutexName)) return FALSE;

    _strcpyW(szBuffer, BASENAMEDOBJECTSDIR);
    _strcatW(szBuffer, MutexName);
    RtlInitUnicodeString(&ustrName, szBuffer);
    InitializeObjectAttributes(&attr, &ustrName, OBJ_CASE_INSENSITIVE, NULL, NULL);
    Status = NtCreateMutant(&hObject, MUTANT_ALL_ACCESS, &attr, FALSE);
    if (NT_SUCCESS(Status)) {
        NtClose(hObject);
        return 0;
    }
    if (Status == STATUS_OBJECT_NAME_COLLISION) return 1;
    return 0;
}
```

FIGURE 2-6. VMDE mutant checker routine.

The same method used to detect Sandboxie presence. Malware checks for

Sandboxie_SingleInstanceMutex_Control

mutex presence. This method can be replaced the bellow described technique. Another way to detect Sandboxie is to check presence of **SbieSvcPort** object inside **\RPC Control** objects directory. You can add here any Sandboxie system object that has name and permanently available. While work Sandboxie allocates new directory in objects directory root called **Sandbox**. It keeps virtualized objects for each sandbox created in Sandboxie. They used to deceive sandboxed applications by replacing global system constants. Note:

the system objects allocated first only when something is running in selected sandbox. Figure 2-7 shows example of Sandboxie directory.

```
lkd> !object \Sandbox
Object: 98628650  Type: (839b4e90) Directory
    ObjectHeader: 98628638 (new version)
    HandleCount: 0  PointerCount: 2
    Directory Object: 87405e28  Name: Sandbox

    Hash Address  Type          Name
    ---- -------- ----          ----
     30  99d85d48 Directory     TestPC        Computer Name
lkd> !object \Sandbox\TestPC
Object: 99d85d48  Type: (839b4e90) Directory
    ObjectHeader: 99d85d30 (new version)
    HandleCount: 0  PointerCount: 4
    Directory Object: 98628650  Name: TestPC

    Hash Address  Type          Name
    ---- -------- ----          ----
     06  99cd9f58 Directory     12345
     11  8783d7b8 Directory     Sandbox_3          Sandbox list
     20  8fa54f58 Directory     DefaultBox
```

FIGURE 2-7. Sandboxie object directory.

To detect presence of Sandboxie is enough simple check if the above-described directory allocated. Note that if Sandboxie was installed but not executed these objects not allocated as well.

```
    if ( IsSB == 0 ) {
        /* not found or error, check sandbox object directory presence */
        RtlInitUnicodeString(&ustrName, DIRECTORY_SANDBOXIE);
        InitializeObjectAttributes(&attr, &ustrName, OBJ_CASE_INSENSITIVE, NULL, NULL);
        Status = NtOpenDirectoryObject(&hObject, DIRECTORY_QUERY, &attr);
        if (NT_SUCCESS(Status)) {
            IsSB = 1;

#ifdef _DEBUG
            DebugLog(TEXT("Sandboxie::ObjectDirectory"));
#endif

            NtClose(hObject);
        }
    }
```

FIGURE 2-8. Detection of Sandboxie object directory.

Malware VM devices hunting usually performed by *CreateFile* Win32 API calls with usage of symbolic links. Disadvantages of this method are the following: you need to keep VM devices symbolic links database, Win32 API may be easily filtered by upper level user mode sandboxing software.

```
if(CreateFile(TEXT("\\\\.\\VBoxMiniRdrDN"),GENERIC_READ,FILE_SHARE_READ,
NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL,NULL) != INVALID_HANDLE_VALUE){
```

FIGURE 2-9. Typical VirtualBox detection by VM Guest Additions used by malware.

How can we change this part? We will switch from *obtaining a handle* of the given device by it link (and thus triggering possible alerts) to *checking presence* of this device object. In other words we will query object via scanning objects directory of the given object type. We decided to use simplified enumerator with callback routine, which will perform objects names comparison. Callback procedure is simple and shown on Figure 2-10.

```
NTSTATUS NTAPI DetectObjectCallback(
    POBJECT_DIRECTORY_INFORMATION Entry,
    PVOID CallbackParam
    )
{
    POBJSCANPARAM Param;

    if (!ARGUMENT_PRESENT(CallbackParam))
        return STATUS_MEMORY_NOT_ALLOCATED;

    Param = (POBJSCANPARAM)CallbackParam;

    __try {
        if (Param->Buffer == NULL || Param->BufferSize == 0)
            return STATUS_MEMORY_NOT_ALLOCATED;

        if (Entry->Name.Buffer) {
            if (_strcmpiW(Entry->Name.Buffer, Param->Buffer) == 0) {
                return STATUS_SUCCESS;
            }
        }
    } __except (EXCEPTION_EXECUTE_HANDLER) {
        /* nothing */
    }
    return STATUS_UNSUCCESSFUL;
}
```

FIGURE 2-10. VMDE query object callback.

Next let us start with enumerator routine, first acquire object directory handle by calling ***NtOpenDirectoryObject***. Notice that as most of Native API of this kind it support supplying not only directory object name, but also a handle to the root directory. Important note: several object directories access requires rights elevation. This applies mostly to "Drivers" directory (as we do not walk all object directories, just a few), but if you have administrator rights this is not a problem. "Device", "BaseNamedObjects" directory in any case will be accessible, otherwise Windows API will fail itself.

```
NTSTATUS NTAPI EnumSystemObjects(
    PWSTR pwszRootDirectory,
    HANDLE hRootDirectory,
    PENUMOBJECTSCALLBACK CallbackProc,
    PVOID CallbackParam
    )
{
    HANDLE hHeap = NULL;
    HANDLE hDirectory = NULL;
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    NTSTATUS CallbackStatus;
    POBJECT_DIRECTORY_INFORMATION pInformation = NULL;
    OBJECT_ATTRIBUTES attr;
    UNICODE_STRING sname;
    BOOL bRootDirectoryNamePresent;

    ULONG k, bytesIO;

    __try {
        if (CallbackProc == NULL) {
            Status = STATUS_NO_CALLBACK_ACTIVE;
            __leave;
        }
        hHeap = RtlProcessHeap();
        bRootDirectoryNamePresent = (pwszRootDirectory != NULL);
        if (bRootDirectoryNamePresent == TRUE) {
            RtlInitUnicodeString(&sname, pwszRootDirectory);
            InitializeObjectAttributes(&attr, &sname, OBJ_CASE_INSENSITIVE, NULL, NULL);
            Status = NtOpenDirectoryObject(&hDirectory, DIRECTORY_QUERY, &attr);
            if (!NT_SUCCESS(Status)) {
                __leave;
            }
        } else {
            hDirectory = hRootDirectory;
        }
```

FIGURE 2-11. VMDE opening objects directory.

Once we have object directory handle we need to walk directory by calling ***NtQueryDirectoryObject*** until it will not return STATUS_NO_MORE_ENTRIES. We use simplified memory allocation, usually it is more than enough but you may consider querying exact size of required memory. Figure 2-12 shows rest of *EnumSystemObjects* routine.

```
        k = 0;
        bytesIO = 0;
        pInformation = (POBJECT_DIRECTORY_INFORMATION)RtlAllocateHeap(hHeap, HEAP_ZERO_MEMORY, PAGE_SIZE * 16);
        if (pInformation != NULL) {
            Status = NtQueryDirectoryObject(hDirectory,
                pInformation, PAGE_SIZE * 16, TRUE, TRUE, &k, &bytesIO);
            if (NT_SUCCESS(Status)) {
                do {
                    CallbackStatus = CallbackProc(pInformation, CallbackParam);
                    if (NT_SUCCESS(CallbackStatus)) {
                        Status = STATUS_SUCCESS;
                        break;
                    }
                    Status = NtQueryDirectoryObject(hDirectory,
                        pInformation, PAGE_SIZE * 16, TRUE, FALSE, &k, &bytesIO);
                } while (Status != STATUS_NO_MORE_ENTRIES);

            }
            RtlFreeHeap(hHeap, 0, pInformation);
        } else {
            Status = STATUS_NO_MEMORY;
            __leave;
        }
    } __finally {
        if (hDirectory != NULL) NtClose(hDirectory);
    }
    return Status;
}
```

FIGURE 2-12. VMDE walking through object directory and calling callback for every entry.

Finally the identification routine, IsObjectExists it is very short and simple, see Figure 2-13.

```
BOOL IsObjectExists(
    PWSTR RootDirectory,
    PWSTR ObjectName
    )
{
    OBJSCANPARAM Param;

    if (!ARGUMENT_PRESENT(ObjectName))
        return FALSE;

    Param.Buffer = ObjectName;
    Param.BufferSize = _strlenW(ObjectName);

    return NT_SUCCESS(EnumSystemObjects(RootDirectory, NULL, DetectObjectCallback, &Param));
}
```

FIGURE 2-13. VMDE IsObjectExists routine.

In the Appendix, you will find examples of this routine usage.

2.2. Emulated Hardware

Every virtual machine provides a set of specially emulated hardware. Usually they are video, network adapters, sound cards or virtual controllers. To be able fully use them user need to install set of VM vendor utilities and drivers (VMware Tools, VirtualBox Guess Additions, Virtual PC VM Additions etc). However even nothing installed, emulated hardware is still here and working. We will abuse this fact and target virtual machine hardware. Usually malware detects suspicious hardware by using documented WMI requests, Setup API or by querying hardware specific registry data without deep knowledge what exactly they are querying. If the virtual machine has been heavily customized all the above tricks will fail.

Some of hardware detection methods implemented relying on fact how VM emulates CPU behavior. For example, using cache instructions, like ***wbinvd*** and ***invd***[15]. Unfortunately, they are privileged[16], which means we cannot use them in user mode and that's break our criteria.

To detect VM presence we will analyze the following data:

- Video BIOS
- System Management BIOS
- PCI bus devices

To get video BIOS and SMB data we will use feature, unavailable in client versions of Windows prior to Vista. Starting from Windows 2003 SP1 as server and Windows Vista RTM as client operation system provides documented[17] way to read system components firmware. We will go little deeper and show you it implementation. Because we are going at internal Native API level, nothing from this documented (and presumably never), everything can be subject of change, so you use this as always on your own risk.

```
typedef enum _SYSTEM_FIRMWARE_TABLE_ACTION
{
  SystemFirmwareTable_Enumerate,
  SystemFirmwareTable_Get
} SYSTEM_FIRMWARE_TABLE_ACTION, *PSYSTEM_FIRMWARE_TABLE_ACTION;

typedef struct _SYSTEM_FIRMWARE_TABLE_INFORMATION {
  ULONG ProviderSignature;
  SYSTEM_FIRMWARE_TABLE_ACTION Action;
  ULONG TableID;
  ULONG TableBufferLength;
  UCHAR TableBuffer[ANYSIZE_ARRAY];
} SYSTEM_FIRMWARE_TABLE_INFORMATION, *PSYSTEM_FIRMWARE_TABLE_INFORMATION;
```

FIGURE 3-1. Structure definitions.

---

[15] http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

[16] http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf

[17] http://msdn.microsoft.com/en-us/library/windows/desktop/ms724379(v=vs.85).aspx

```
PVOID GetFirmwareTable(
    PULONG pdwDataSize,
    DWORD dwSignature,
    DWORD dwTableID
    )
{
    NTSTATUS Status;
    ULONG Length;
    HANDLE hProcess = NULL;
    ULONG uAddress;
    PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;
    SIZE_T memIO = 0;

    CLIENT_ID cid;
    OBJECT_ATTRIBUTES attr;
    MEMORY_REGION_INFORMATION memInfo;

    /* use documented GetSystemFirmwareTable instead, this is it raw implementation */
    if ( g_osVer.dwMajorVersion > 5 ) {
        Length = 0x1000;
        pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY, Length);
        if (pSIF != NULL) {
            pSIF->Action = SystemFirmwareTable_Get;
            pSIF->ProviderSignature = dwSignature;
            pSIF->TableID = dwTableID;
            pSIF->TableBufferLength = Length;
            /* query if info class available and if how many memory we need  */
            Status = NtQuerySystemInformation(SystemFirmwareTableInformation, pSIF, Length, &Length);
            if (Status == STATUS_INVALID_INFO_CLASS || Status == STATUS_INVALID_DEVICE_REQUEST) {
                RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                return NULL;
            }
            if (!NT_SUCCESS(Status) || Status == STATUS_BUFFER_TOO_SMALL) {
                RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY, Length);
                if (pSIF != NULL) {
                    pSIF->Action = SystemFirmwareTable_Get;
                    pSIF->ProviderSignature = dwSignature;
                    pSIF->TableID = dwTableID;
                    pSIF->TableBufferLength = Length;
                    Status = NtQuerySystemInformation(SystemFirmwareTableInformation, pSIF, Length, &Length);
                    if (!NT_SUCCESS(Status)) {
                        RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                        return NULL;
                    }
                    if (ARGUMENT_PRESENT(pdwDataSize))
                        *pdwDataSize = Length;
                }
            } else {
                if (ARGUMENT_PRESENT(pdwDataSize))
                    *pdwDataSize = Length;
            }
        }
    } else {
```

FIGURE 3-2. VMDE GetFirmwareTable implementation.

However, what to do with legacy Windows versions such as almost dead Windows XP SP3? On Windows XP SP3 the above info class unavailable, but all required information located inside Client/Server Runtime Subsystem (CSRSS*)* memory space, stored here for Virtual DOS Machine (VDM) purposes, at few fixed addresses. Access to the csrss process requires *SeDebugPrivilege* enabled for caller process. That is a restriction, but since all Windows XP systems by default running everything with admin rights, it is does not matter in this case. Figure 3-3 shows memory map of csrss with selected memory region that hold Video BIOS data.

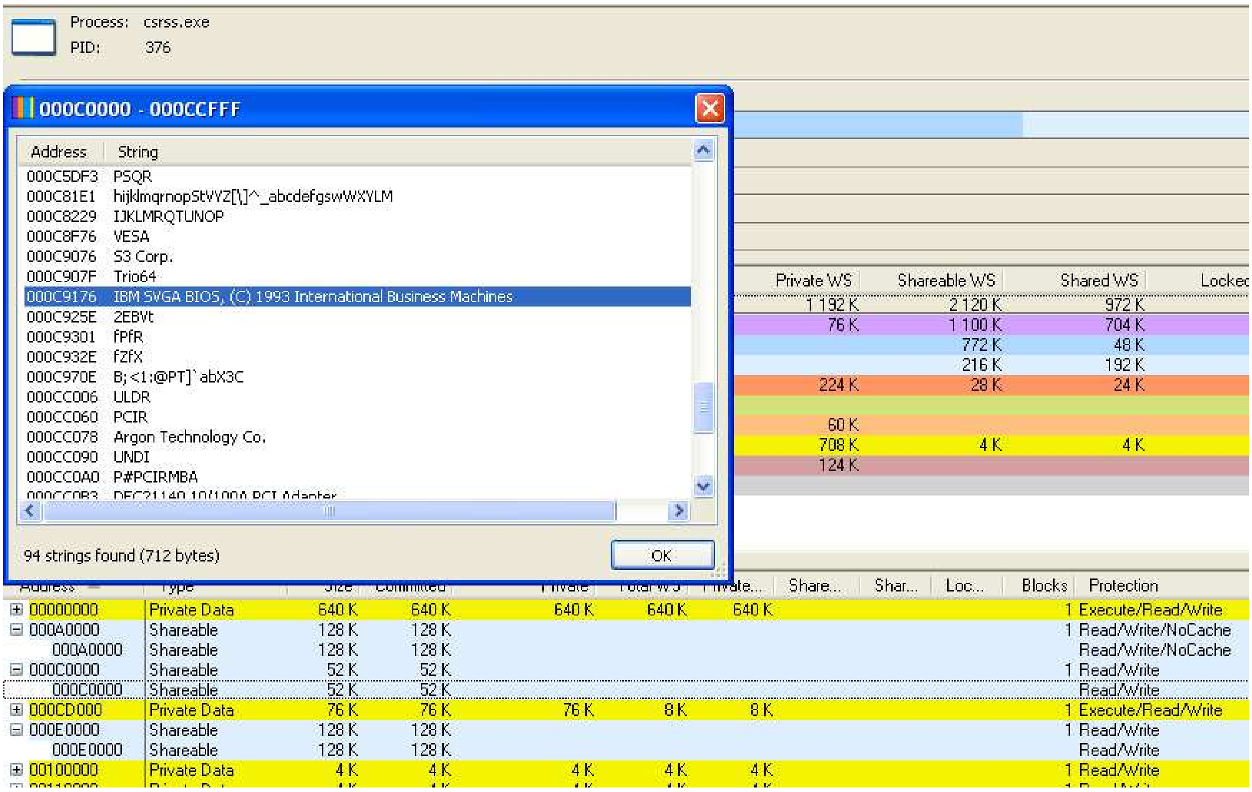

FIGURE 3-3. Video BIOS data inside csrss memory, Windows XP SP3.

Memory region with address 0xC0000 holds raw firmware data, and memory region with address 0xE0000 holds raw SMBIOS data. Dump it from there as shown on Figure 3-4. After dumping contents of these tables all is left run signature scanning, locating VM specific data. Note that Microsoft Virtual PC video card is emulated S3 Trio64 adapter with complete S3 Trio64 VGA BIOS. Ironically, only fact of old S3 Graphics based adapter is a good indication of virtual machine.

```
    } else {
        /* on pre Vista systems the above info class unavailable, but all required information
           can be found inside csrss  memory space (stored here for VMD purposes) at few fixed addresses
        */
        if ((dwSignature != FIRM) && (dwSignature != RSMB)) return NULL;

        /* we are interested only in two memory regions */
        switch ( dwTableID ) {
        case FIRM:
            uAddress = 0xC0000; /* FIRM analogue */
            break;
        case RSMB:
            uAddress = 0xE0000; /* RSMB analogue */
            break;
        default:
            return NULL;
            break;
        }

        Length = 0;
        cid.UniqueProcess = (HANDLE)CsrGetProcessId();
        cid.UniqueThread = 0;
        InitializeObjectAttributes(&attr, NULL, 0, 0, NULL);
        /* open csrss, reg. client debug privilege set */
        Status = NtOpenProcess(&hProcess, PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, &attr, &cid);
        if (NT_SUCCESS(Status)) {
            /* get memory data region size for buffer allocation */
            Status = NtQueryVirtualMemory(hProcess, (PVOID)uAddress, MemoryRegionInformation, &memInfo, sizeof(MEMORY_REGION_INFORMATION), &memIO);
            if (NT_SUCCESS(Status)) {
                pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY, memInfo.RegionSize);
                if (pSIF != NULL) {
                    /* read data to our allocated buffer */
                    Status = NtReadVirtualMemory(hProcess, (PVOID)uAddress, pSIF, memInfo.RegionSize, &memIO);
                    if (NT_SUCCESS(Status)) {
                        if (ARGUMENT_PRESENT(pdwDataSize))
                            *pdwDataSize = memInfo.RegionSize;
                    } else {
                        RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                        return NULL;
                    }
                }
            }
            NtClose(hProcess);
        }
    }
    return pSIF;
}
```

FIGURE 3-4. The rest of VMDE GetFirmwareTable.

Next few figures demonstrate contents of firmware data used by popular virtual machines.

FIGURE 3-5. Part of Virtual PC firmware dump, S3 VGA BIOS.

FIGURE 3-6. Part of Virtual PC RSMB dump.

FIGURE 3-7. Part of VMware firmware dump.

FIGURE 3-8. Part of VMware RSMB dump.

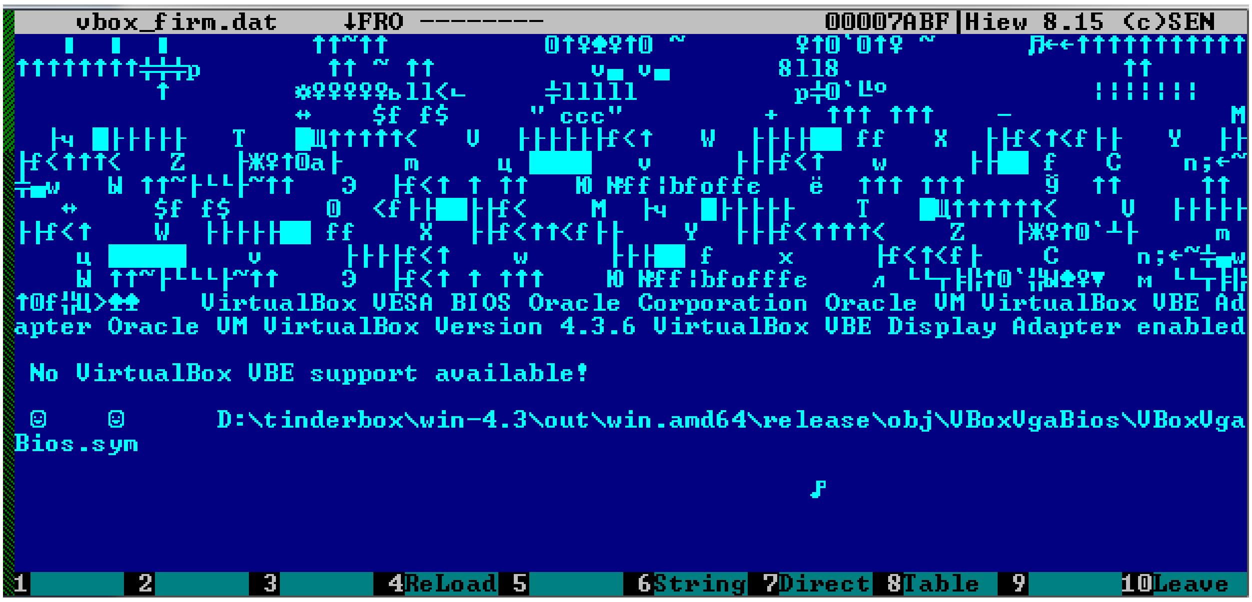

FIGURE 3-9. Part of Virtual Box firmware dump.

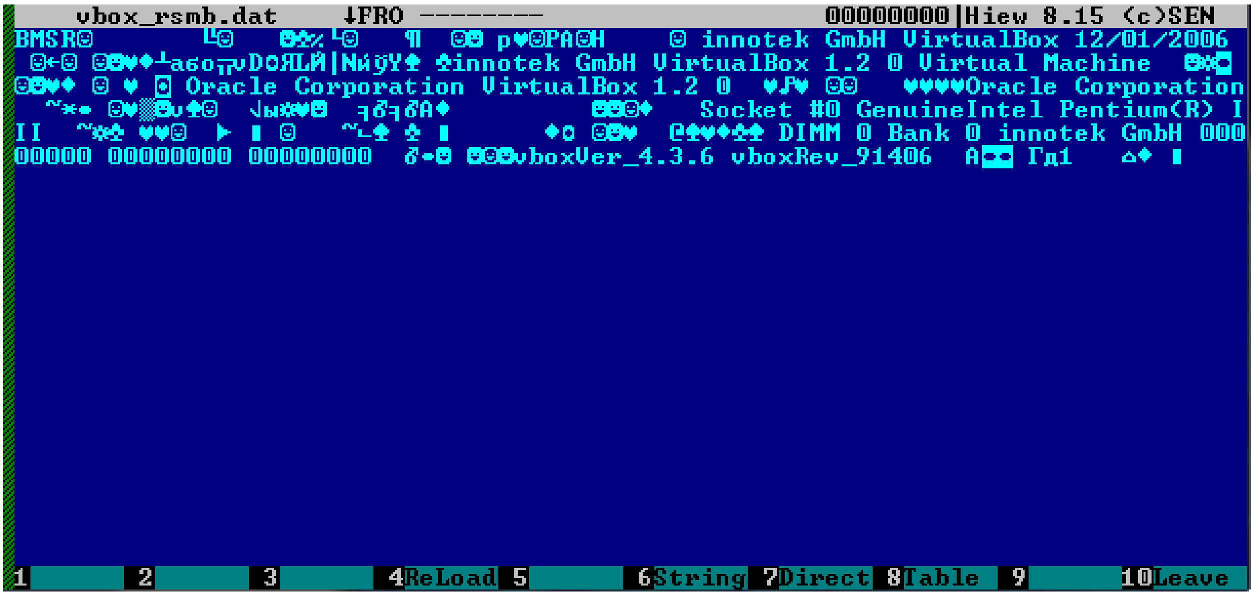

FIGURE 3-10. Part of Virtual Box RSMB dump.

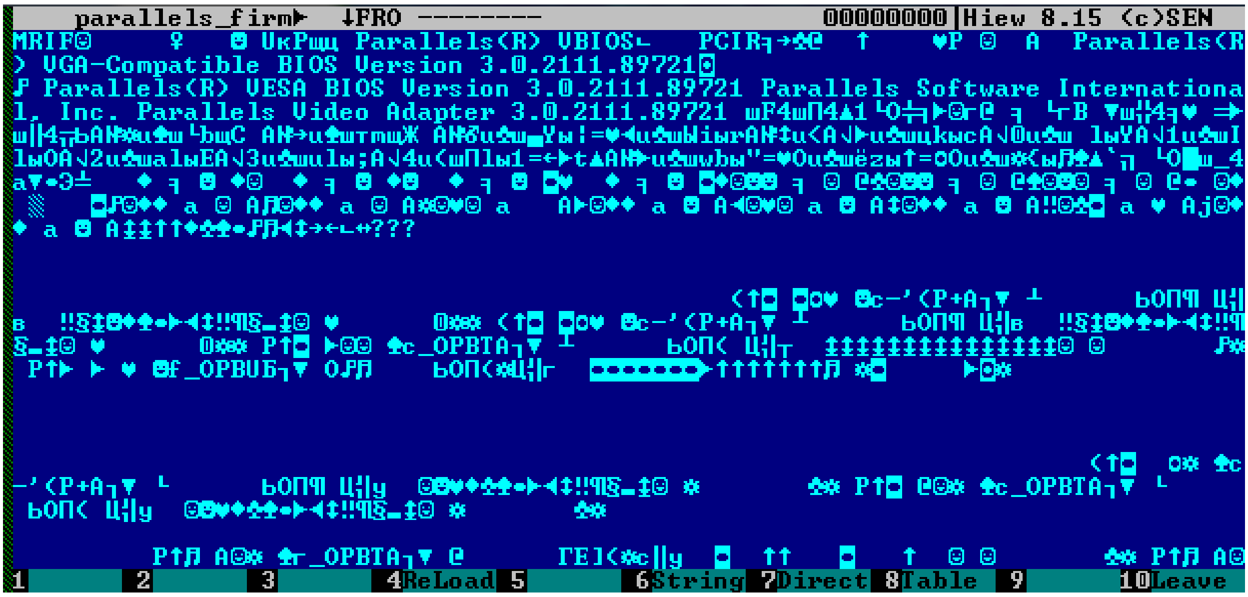

FIGURE 3-11. Part of Parallels Desktop firmware dump.

FIGURE 3-12. Part of Parallels RSMB dump.

This information is not depending on VM tools installation but as you might guess, heavy manually customized VM can bypass that check, but not all VM may be reconfigured that way without execution problem. Another way to detect VM presence is scanning PCI bus devices. Figure 3-13 shows PCI bus connected devices on the Virtual PC.

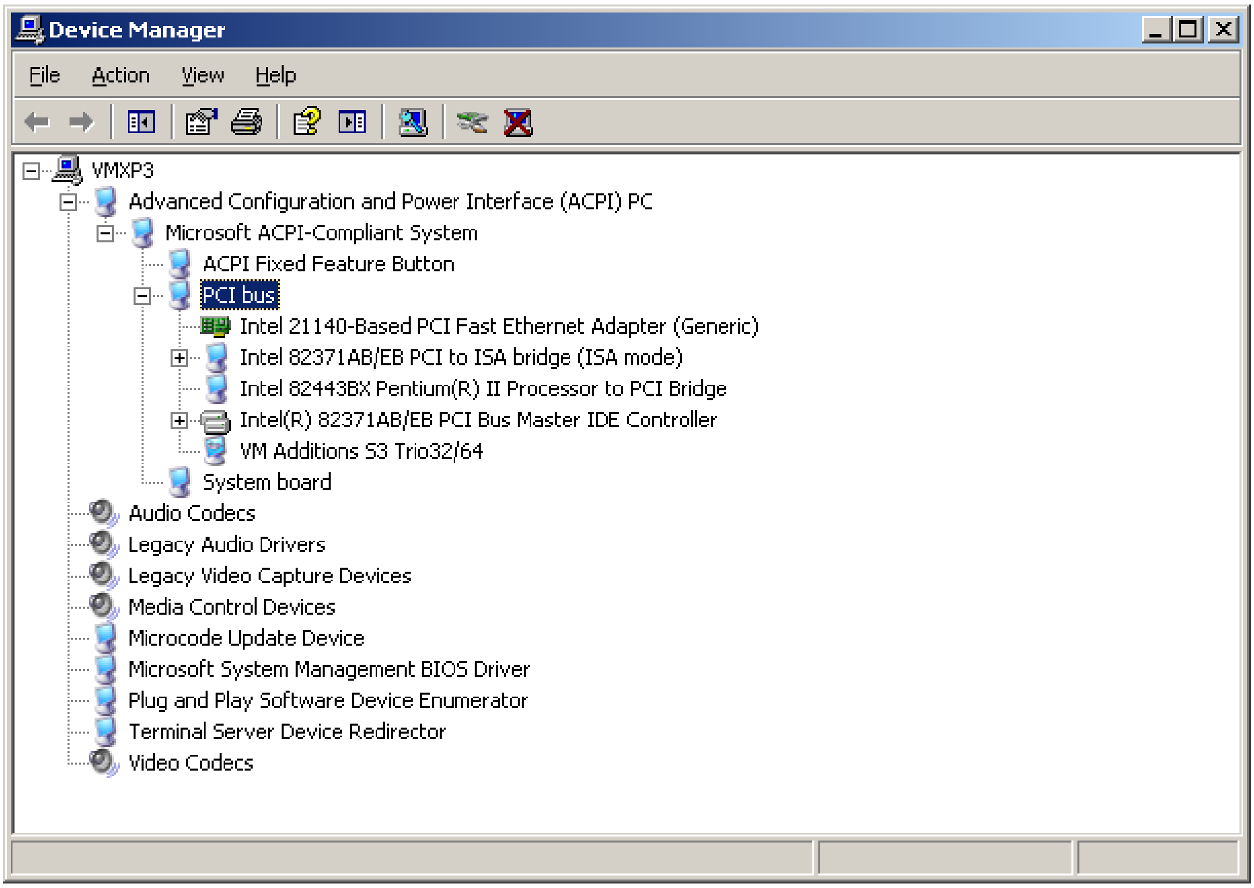

FIGURE 3-13. PCI bus connected devices.

Usually malware does not take this information in credit and only scans registry for specific vendor names like *VBOX*, *VMWARE* and so on. We will detect VM presence in a different way by enumerating PCI bus devices and querying their Hardware vendor ID (HWID). The problem is again that we must do this from user mode and if possible, without elevating privileges, so we cannot use *SeTcbPrivilege* and *ProcessUserModeIOPL* to list PCI devices manually. Instead, we will use system information stored inside

\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Enum\PCI

We even do not need to read anything from it, it is enough to simple enumerate the root key. Each key here represent a PCI bus connected device and has the following device instance format:

VEN_XXXX&DEV_XXXX&SUBSYS_XXXXXXXX&REV_XX

Where VEN stands for Vendor ID in hexadecimal view, DEV stands for Device ID in hexadecimal view. While dumping content of the root key we will process each entry and extract VID value from it. See Appendix for a examples of usage.

```
VOID EnumPCIDevsReg(
    VOID
    )
{
    HANDLE hKey = NULL;
    DWORD dwKeySubIndex = 0;
    ULONG ResultLength = 0;
    NTSTATUS Status = STATUS_UNSUCCESSFUL;

    UNICODE_STRING ustrKeyName;
    OBJECT_ATTRIBUTES obja;
    PKEY_BASIC_INFORMATION pKeyInfo = NULL;

    PVENDOR_ENTRY entry;
    WCHAR szTempBuf[MAX_PATH] = {0};

    RtlInitUnicodeString(&ustrKeyName, REGSTR_KEY_PCIENUM);
    InitializeObjectAttributes(&obja, &ustrKeyName, OBJ_CASE_INSENSITIVE, NULL, NULL);

    __try {
        Status = NtOpenKey(&hKey, KEY_ENUMERATE_SUB_KEYS, &obja);
        if ( hKey == NULL && !NT_SUCCESS(Status)) __leave;
        do {

            NtEnumerateKey(hKey, dwKeySubIndex, KeyBasicInformation, NULL, 0, &ResultLength);
            pKeyInfo = (PKEY_BASIC_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY, ResultLength);
            if ( pKeyInfo == NULL) __leave;

            Status = NtEnumerateKey(hKey, dwKeySubIndex, KeyBasicInformation, pKeyInfo, ResultLength, &ResultLength);
            if (NT_SUCCESS(Status)) {
                entry = (PVENDOR_ENTRY)mmalloc(sizeof(VENDOR_ENTRY));
                if ( entry ) {
                    _strncpyW(entry->VendorFullName, MAX_PATH, pKeyInfo->Name, pKeyInfo->NameLength / sizeof(WCHAR));
                    vExtractID(entry);
                    InsertHeadList(&VendorsListHead, &entry->ListEntry);
                }
                RtlFreeHeap(RtlProcessHeap(), 0, pKeyInfo);
                pKeyInfo = NULL;
            }
            dwKeySubIndex++;

        } while ( NT_SUCCESS(Status) );

    } __finally {
        if (hKey != NULL) NtClose(hKey);
        if ( pKeyInfo != NULL) RtlFreeHeap(RtlProcessHeap(), 0, pKeyInfo);
    }
}
```

FIGURE 3-14. VMDE HWID enumeration routine.

Once we know all PCI bus devices we can walk through ID table and compare with known VM vendors ID. Table 3-1 contains list of known vendors.

TABLE 3-1. Virtual Machine known hardware vendor ID.

Vendor	Vendor ID
VMware	0x15AD
Oracle	0x80EE
Parallels	0x1AB8

In case of Virtual PC you can query S3 VID which is 0x5333, but take care as this can trigger false positives.

2.3. Additional execution artifacts.

While running certain virtualization software always produces artifacts that cannot occur outside VM/Sandbox environment. They can be different: mismatches between hardware information returned by CPU and operation system (number of cores, suspicious CPU vendor names), incorrectly emulated instructions, traces of virtualization software code execution. The perfect example of these artifacts, we can say artifacts generator, is Sandboxie. Some of it detection methods already mentioned in previous chapters and in-the-wild malware uses only few of them. What Sandboxie actually does:

- Isolates selected application from the rest of the system by heavy kernel mode modifications to process affiliated system structures;
- Provides transparent layer of API call redirects and filtering, controlling application execution behavior.

How secure this compared to execution on the VM? The both sandbox and virtual machine might have many in common goals but very different in implementation and result. Therefore, we would say they could not be compared because code runs at different conditions. However, instead of pure VM Sandboxie will always suffer from critical implementation bugs[18] that may alter behavior of real system and/or even compromise it security. We did not say that VM is free from implementation bugs[19] [20], but their usage and effect is incomparable to bugs in the sandbox software that shares same host resources and session. In other words, sandboxing software is mostly for casual users, while virtual machines are mode advanced and more user skill dependent. Here we must distinguish sandboxing *software* oriented for end-users and sandboxing as *implementation* used for example in antivirus products and laboratories.

We will abuse Sandboxie transparent API layer to detect if our application is being sandboxed. There are multiple detection ways, but we will show only most obvious. One of the ways is that how Sandboxie implements program isolation through virtualized registry. Big role here plays helper library SBIEDLL.DLL that injected in every sandboxed process. This dll itself is a great indication of Sandboxie and this fact widely used by malware. Nevertheless, exist helper applications that hide presence of this dll[21]. Virtualized registry only present in sandboxed applications so all we need is to establish fact of such virtualization. Each sandbox has its own dedicated fake registry entries (just like in case of system named objects). Figure 3-15 shows example.

---

[18] http://www.kernelmode.info/forum/viewtopic.php?f=13&t=2244

[19] https://www.virtualbox.org/ticket/10947

[20] http://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-20406/Oracle-Vm-Virtualbox.html

[21] http://bsa.isoftware.nl/

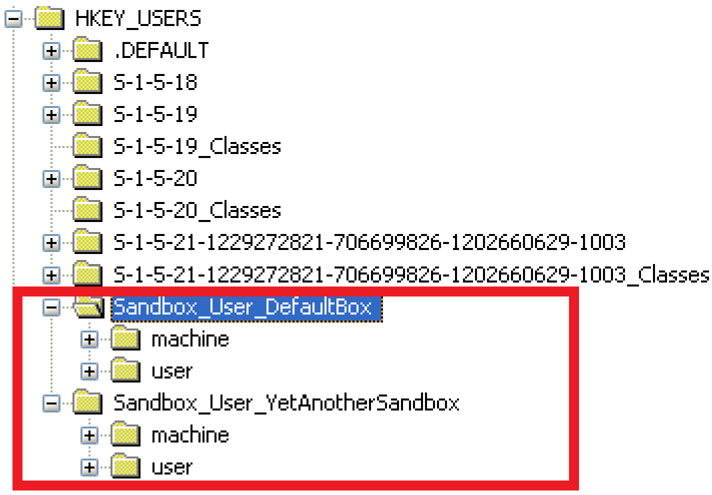

FIGURE 3-15. Sandboxie registry virtualization.

To make this virtualization transparent to the sandboxed application SBIEDLL.DLL component perform massive API filtering in the user mode, faking results of several APIs. Below is list of massive code modifications found in the sandboxed process running under Windows XP SP3. Note that under different platforms this list may look different. These standard libraries modifications itself by the way can be used to detect sandboxing.

```
ntdll.dll-->LdrInitializeThunk Inline - RelativeJump 0x7C901166-->7E420008 [unknown_code_page]
ntdll.dll-->LdrLoadDll Inline - RelativeJump 0x7C9163A3-->7D2538E0 [SbieDll.dll]
ntdll.dll-->LdrQueryImageFileExecutionOptions Inline - RelativeJump 0x7C91CC83-->7D2539E0
[SbieDll.dll]
ntdll.dll-->LdrUnloadDll Inline - RelativeJump 0x7C91736B-->7D253940 [SbieDll.dll]
ntdll.dll-->NtAcceptConnectPort Code Mismatch 0x7C90CE40 + 4 [18 E9 16 32 B1 01]
ntdll.dll-->NtAccessCheck Code Mismatch 0x7C90CE50 + 4 [20 E9 06 32 B1 01]
ntdll.dll-->NtAccessCheckAndAuditAlarm Code Mismatch 0x7C90CE60 + 4 [2C E9 F6 31 B1 01]
ntdll.dll-->NtAccessCheckByType Code Mismatch 0x7C90CE70 + 4 [2C E9 E6 31 B1 01]
ntdll.dll-->NtAccessCheckByTypeAndAuditAlarm Inline - RelativeJump 0x7C90CE84-->7E420060
[unknown_code_page]
ntdll.dll-->NtAccessCheckByTypeResultList Code Mismatch 0x7C90CE90 + 4 [2C E9 C6 31 B1 01]
ntdll.dll-->NtAccessCheckByTypeResultListAndAuditAlarm Inline - RelativeJump 0x7C90CEA4--
>7E420060 [unknown_code_page]
ntdll.dll-->NtAccessCheckByTypeResultListAndAuditAlarmByHandle Inline - RelativeJump 0x7C90CEB4--
>7E420060 [unknown_code_page]
ntdll.dll-->NtAddAtom Code Mismatch 0x7C90CEC0 + 4 [0C E9 96 31 B1 01]
ntdll.dll-->NtAddBootEntry Code Mismatch 0x7C90CED0 + 4 [08 E9 86 31 B1 01]
ntdll.dll-->NtAdjustGroupsToken Inline - RelativeJump 0x7C90CEE4-->7C90CF1D [ntdll.dll]
ntdll.dll-->NtAdjustPrivilegesToken Inline - RelativeJump 0x7C90CEF0-->7D25E2F0 [SbieDll.dll]
ntdll.dll-->NtAdjustPrivilegesToken Inline - RelativeJump 0x7C90CEF5-->7E420060
[unknown_code_page]
ntdll.dll-->NtAlertResumeThread Code Mismatch 0x7C90CF00 + 4 [08 E9 56 31 B1 01]
ntdll.dll-->NtAlertThread Code Mismatch 0x7C90CF10 + 4 [04 E9 46 31 B1 01]
ntdll.dll-->NtAllocateLocallyUniqueId Code Mismatch 0x7C90CF20 + 4 [04 E9 36 31 B1 01]
ntdll.dll-->NtAllocateUserPhysicalPages Code Mismatch 0x7C90CF30 + 4 [0C E9 26 31 B1 01]
ntdll.dll-->NtAllocateUuids Code Mismatch 0x7C90CF40 + 4 [10 E9 16 31 B1 01]
ntdll.dll-->NtAllocateVirtualMemory Code Mismatch 0x7C90CF50 + 4 [18 E9 06 31 B1 01]
ntdll.dll-->NtAreMappedFilesTheSame Code Mismatch 0x7C90CF60 + 4 [08 E9 F6 30 B1 01]
ntdll.dll-->NtAssignProcessToJobObject Inline - RelativeJump 0x7C90CF70-->7D260930 [SbieDll.dll]
ntdll.dll-->NtAssignProcessToJobObject Inline - RelativeJump 0x7C90CF75-->7E420060
[unknown_code_page]
ntdll.dll-->NtCancelDeviceWakeupRequest Code Mismatch 0x7C90CF90 + 4 [04 E9 C6 30 B1 01]
ntdll.dll-->NtCancelIoFile Code Mismatch 0x7C90CFA0 + 4 [08 E9 B6 30 B1 01]
ntdll.dll-->NtCancelTimer Code Mismatch 0x7C90CFB0 + 4 [08 E9 A6 30 B1 01]
ntdll.dll-->NtClearEvent Code Mismatch 0x7C90CFC0 + 4 [04 E9 96 30 B1 01]
ntdll.dll-->NtClose Inline - RelativeJump 0x7C90CFD0-->7D23CC70 [SbieDll.dll]
ntdll.dll-->NtClose Inline - RelativeJump 0x7C90CFD5-->7E420060 [unknown_code_page]
ntdll.dll-->NtCloseObjectAuditAlarm Inline - RelativeJump 0x7C90CFE4-->7C90D01C [ntdll.dll]
ntdll.dll-->NtCompactKeys Code Mismatch 0x7C90CFF0 + 4 [08 E9 66 30 B1 01]
ntdll.dll-->NtCompareTokens Code Mismatch 0x7C90D000 + 4 [0C E9 56 30 B1 01]
```

```
ntdll.dll-->NtCompleteConnectPort Code Mismatch 0x7C90D010 + 4 [04 E9 46 30 B1 01]
ntdll.dll-->NtCompressKey Code Mismatch 0x7C90D020 + 4 [04 E9 36 30 B1 01]
ntdll.dll-->NtConnectPort Inline - RelativeJump 0x7C90D030-->7D24BEC0 [SbieDll.dll]
ntdll.dll-->NtConnectPort Inline - RelativeJump 0x7C90D035-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateDebugObject Code Mismatch 0x7C90D050 + 4 [10 E9 06 30 B1 01]
ntdll.dll-->NtCreateDirectoryObject Code Mismatch 0x7C90D060 + 4 [0C E9 F6 2F B1 01]
ntdll.dll-->NtCreateEvent Inline - RelativeJump 0x7C90D070-->7D24C9B0 [SbieDll.dll]
ntdll.dll-->NtCreateEvent Inline - RelativeJump 0x7C90D075-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateEventPair Code Mismatch 0x7C90D080 + 4 [0C E9 D6 2F B1 01]
ntdll.dll-->NtCreateFile Inline - RelativeJump 0x7C90D090-->7D23D4D0 [SbieDll.dll]
ntdll.dll-->NtCreateFile Inline - RelativeJump 0x7C90D095-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateIoCompletion Code Mismatch 0x7C90D0A0 + 4 [10 E9 B6 2F B1 01]
ntdll.dll-->NtCreateJobObject Inline - RelativeJump 0x7C90D0B0-->7D260700 [SbieDll.dll]
ntdll.dll-->NtCreateJobObject Inline - RelativeJump 0x7C90D0B5-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateJobSet Code Mismatch 0x7C90D0C0 + 4 [0C E9 96 2F B1 01]
ntdll.dll-->NtCreateKey Inline - RelativeJump 0x7C90D0D0-->7D251AC0 [SbieDll.dll]
ntdll.dll-->NtCreateKey Inline - RelativeJump 0x7C90D0D5-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateKeyedEvent Code Mismatch 0x7C90DFB0 + 4 [10 E9 A6 20 B1 01]
ntdll.dll-->NtCreateMailslotFile Inline - RelativeJump 0x7C90D0E0-->7D23A3B0 [SbieDll.dll]
ntdll.dll-->NtCreateMailslotFile Inline - RelativeJump 0x7C90D0E5-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateMutant Inline - RelativeJump 0x7C90D0F0-->7D24CD10 [SbieDll.dll]
ntdll.dll-->NtCreateMutant Inline - RelativeJump 0x7C90D0F5-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateNamedPipeFile Inline - RelativeJump 0x7C90D100-->7D23A550 [SbieDll.dll]
ntdll.dll-->NtCreateNamedPipeFile Inline - RelativeJump 0x7C90D105-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreatePagingFile Code Mismatch 0x7C90D110 + 4 [10 E9 46 2F B1 01]
ntdll.dll-->NtCreatePort Inline - RelativeJump 0x7C90D120-->7D24BD50 [SbieDll.dll]
ntdll.dll-->NtCreatePort Inline - RelativeJump 0x7C90D125-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateProcess Code Mismatch 0x7C90D130 + 4 [20 E9 26 2F B1 01]
ntdll.dll-->NtCreateProcessEx Code Mismatch 0x7C90D140 + 4 [24 E9 16 2F B1 01]
ntdll.dll-->NtCreateProfile Code Mismatch 0x7C90D150 + 4 [24 E9 06 2F B1 01]
ntdll.dll-->NtCreateSection Inline - RelativeJump 0x7C90D160-->7D24D3A0 [SbieDll.dll]
ntdll.dll-->NtCreateSection Inline - RelativeJump 0x7C90D165-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateSemaphore Inline - RelativeJump 0x7C90D170-->7D24D040 [SbieDll.dll]
ntdll.dll-->NtCreateSemaphore Inline - RelativeJump 0x7C90D175-->7E420060 [unknown_code_page]
ntdll.dll-->NtCreateSymbolicLinkObject Code Mismatch 0x7C90D180 + 4 [10 E9 D6 2E B1 01]
ntdll.dll-->NtCreateThread Code Mismatch 0x7C90D190 + 4 [20 E9 C6 2E B1 01]
ntdll.dll-->NtCreateTimer Code Mismatch 0x7C90D1A0 + 4 [10 E9 B6 2E B1 01]
ntdll.dll-->NtCreateToken Code Mismatch 0x7C90D1B0 + 4 [34 E9 A6 2E B1 01]
ntdll.dll-->NtCreateWaitablePort Code Mismatch 0x7C90D1C0 + 4 [14 E9 96 2E B1 01]
ntdll.dll-->NtDebugActiveProcess Code Mismatch 0x7C90D1D0 + 4 [08 E9 86 2E B1 01]
ntdll.dll-->NtDebugContinue Inline - RelativeJump 0x7C90D1E4-->7C90D21A [ntdll.dll]
ntdll.dll-->NtDelayExecution Code Mismatch 0x7C90D1F0 + 4 [08 E9 66 2E B1 01]
ntdll.dll-->NtDeleteAtom Code Mismatch 0x7C90D200 + 4 [04 E9 56 2E B1 01]
ntdll.dll-->NtDeleteBootEntry Code Mismatch 0x7C90D210 + 4 [04 E9 46 2E B1 01]
ntdll.dll-->NtDeleteFile Inline - RelativeJump 0x7C90D220-->7D235BD0 [SbieDll.dll]
ntdll.dll-->NtDeleteFile Inline - RelativeJump 0x7C90D225-->7E420060 [unknown_code_page]
ntdll.dll-->NtDeleteKey Inline - RelativeJump 0x7C90D230-->7D252CA0 [SbieDll.dll]
ntdll.dll-->NtDeleteKey Inline - RelativeJump 0x7C90D235-->7E420060 [unknown_code_page]
ntdll.dll-->NtDeleteObjectAuditAlarm Code Mismatch 0x7C90D240 + 4 [0C E9 16 2E B1 01]
ntdll.dll-->NtDeleteValueKey Inline - RelativeJump 0x7C90D250-->7D252B30 [SbieDll.dll]
ntdll.dll-->NtDeleteValueKey Inline - RelativeJump 0x7C90D255-->7E420060 [unknown_code_page]
ntdll.dll-->NtDeviceIoControlFile Inline - RelativeJump 0x7C90D260-->7D23A800 [SbieDll.dll]
ntdll.dll-->NtDeviceIoControlFile Inline - RelativeJump 0x7C90D265-->7E420060 [unknown_code_page]
ntdll.dll-->NtDisplayString Code Mismatch 0x7C90D270 + 4 [04 E9 E6 2D B1 01]
ntdll.dll-->NtDuplicateObject Inline - RelativeJump 0x7C90D280-->7D25EA40 [SbieDll.dll]
ntdll.dll-->NtDuplicateObject Inline - RelativeJump 0x7C90D285-->7E420060 [unknown_code_page]
ntdll.dll-->NtDuplicateToken Code Mismatch 0x7C90D290 + 4 [18 E9 C6 2D B1 01]
ntdll.dll-->NtEnumerateBootEntries Code Mismatch 0x7C90D2A0 + 4 [08 E9 B6 2D B1 01]
ntdll.dll-->NtEnumerateKey Inline - RelativeJump 0x7C90D2B0-->7D251E40 [SbieDll.dll]
ntdll.dll-->NtEnumerateKey Inline - RelativeJump 0x7C90D2B5-->7E420060 [unknown_code_page]
ntdll.dll-->NtEnumerateSystemEnvironmentValuesEx Code Mismatch 0x7C90D2C0 + 4 [0C E9 96 2D B1 01]
ntdll.dll-->NtEnumerateValueKey Inline - RelativeJump 0x7C90D2D0-->7D252490 [SbieDll.dll]
ntdll.dll-->NtEnumerateValueKey Inline - RelativeJump 0x7C90D2D5-->7E420060 [unknown_code_page]
ntdll.dll-->NtExtendSection Inline - RelativeJump 0x7C90D2E4-->7C90D319 [ntdll.dll]
ntdll.dll-->NtFilterToken Code Mismatch 0x7C90D2F0 + 4 [18 E9 66 2D B1 01]
ntdll.dll-->NtFindAtom Code Mismatch 0x7C90D300 + 4 [0C E9 56 2D B1 01]
ntdll.dll-->NtFlushBuffersFile Code Mismatch 0x7C90D310 + 4 [08 E9 46 2D B1 01]
ntdll.dll-->NtFlushInstructionCache Code Mismatch 0x7C90D320 + 4 [0C E9 36 2D B1 01]
ntdll.dll-->NtFlushKey Code Mismatch 0x7C90D330 + 4 [04 E9 26 2D B1 01]
ntdll.dll-->NtFlushVirtualMemory Code Mismatch 0x7C90D340 + 4 [10 E9 16 2D B1 01]
ntdll.dll-->NtFlushWriteBuffer Inline - RelativeJump 0x7C90D355-->7E420060 [unknown_code_page]
ntdll.dll-->NtFreeUserPhysicalPages Code Mismatch 0x7C90D360 + 4 [0C E9 F6 2C B1 01]
ntdll.dll-->NtFreeVirtualMemory Code Mismatch 0x7C90D370 + 4 [10 E9 E6 2C B1 01]
ntdll.dll-->NtFsControlFile Inline - RelativeJump 0x7C90D380-->7D23CAD0 [SbieDll.dll]
ntdll.dll-->NtFsControlFile Inline - RelativeJump 0x7C90D385-->7E420060 [unknown_code_page]
ntdll.dll-->NtGetContextThread Code Mismatch 0x7C90D390 + 4 [08 E9 C6 2C B1 01]
```

```
ntdll.dll-->NtGetDevicePowerState Code Mismatch 0x7C90D3A0 + 4 [08 E9 B6 2C B1 01]
ntdll.dll-->NtGetPlugPlayEvent Code Mismatch 0x7C90D3B0 + 4 [10 E9 A6 2C B1 01]
ntdll.dll-->NtGetWriteWatch Code Mismatch 0x7C90D3C0 + 4 [1C E9 96 2C B1 01]
ntdll.dll-->NtImpersonateAnonymousToken Inline - RelativeJump 0x7C90D3D0-->7D24ACA0 [SbieDll.dll]
ntdll.dll-->NtImpersonateAnonymousToken Inline - RelativeJump 0x7C90D3D5-->7E420060
[unknown_code_page]
ntdll.dll-->NtImpersonateClientOfPort Inline - RelativeJump 0x7C90D3E0-->7D24AC30 [SbieDll.dll]
ntdll.dll-->NtImpersonateClientOfPort Inline - RelativeJump 0x7C90D3E5-->7E420060
[unknown_code_page]
ntdll.dll-->NtImpersonateThread Inline - RelativeJump 0x7C90D3F0-->7D24ACE0 [SbieDll.dll]
ntdll.dll-->NtImpersonateThread Inline - RelativeJump 0x7C90D3F5-->7E420060 [unknown_code_page]
ntdll.dll-->NtInitializeRegistry Code Mismatch 0x7C90D400 + 4 [04 E9 56 2C B1 01]
ntdll.dll-->NtInitiatePowerAction Code Mismatch 0x7C90D410 + 4 [10 E9 46 2C B1 01]
ntdll.dll-->NtIsProcessInJob Code Mismatch 0x7C90D420 + 4 [08 E9 36 2C B1 01]
ntdll.dll-->NtIsSystemResumeAutomatic Inline - RelativeJump 0x7C90D435-->7E420060
[unknown_code_page]
ntdll.dll-->NtListenPort Code Mismatch 0x7C90D440 + 4 [08 E9 16 2C B1 01]
ntdll.dll-->NtLoadDriver Inline - RelativeJump 0x7C90D450-->7D253BF0 [SbieDll.dll]
ntdll.dll-->NtLoadDriver Inline - RelativeJump 0x7C90D455-->7E420060 [unknown_code_page]
ntdll.dll-->NtLoadKey Inline - RelativeJump 0x7C90D460-->7D250370 [SbieDll.dll]
ntdll.dll-->NtLoadKey Inline - RelativeJump 0x7C90D465-->7E420060 [unknown_code_page]
ntdll.dll-->NtLoadKey2 Code Mismatch 0x7C90D470 + 4 [0C E9 E6 2B B1 01]
ntdll.dll-->NtLockFile Code Mismatch 0x7C90D480 + 4 [28 E9 D6 2B B1 01]
ntdll.dll-->NtLockProductActivationKeys Code Mismatch 0x7C90D490 + 4 [08 E9 C6 2B B1 01]
ntdll.dll-->NtLockRegistryKey Code Mismatch 0x7C90D4A0 + 4 [04 E9 B6 2B B1 01]
ntdll.dll-->NtLockVirtualMemory Code Mismatch 0x7C90D4B0 + 4 [10 E9 A6 2B B1 01]
ntdll.dll-->NtMakePermanentObject Code Mismatch 0x7C90D4C0 + 4 [04 E9 96 2B B1 01]
ntdll.dll-->NtMakeTemporaryObject Code Mismatch 0x7C90D4D0 + 4 [04 E9 86 2B B1 01]
ntdll.dll-->NtMapUserPhysicalPages Inline - RelativeJump 0x7C90D4E4-->7C90D517 [ntdll.dll]
ntdll.dll-->NtMapUserPhysicalPagesScatter Code Mismatch 0x7C90D4F0 + 4 [0C E9 66 2B B1 01]
ntdll.dll-->NtModifyBootEntry Code Mismatch 0x7C90D510 + 4 [04 E9 46 2B B1 01]
ntdll.dll-->NtNotifyChangeDirectoryFile Code Mismatch 0x7C90D520 + 4 [24 E9 36 2B B1 01]
ntdll.dll-->NtNotifyChangeKey Inline - RelativeJump 0x7C90D530-->7D250DD0 [SbieDll.dll]
ntdll.dll-->NtNotifyChangeKey Inline - RelativeJump 0x7C90D535-->7E420060 [unknown_code_page]
ntdll.dll-->NtNotifyChangeMultipleKeys Inline - RelativeJump 0x7C90D540-->7D24FFB0 [SbieDll.dll]
ntdll.dll-->NtNotifyChangeMultipleKeys Inline - RelativeJump 0x7C90D545-->7E420060
[unknown_code_page]
ntdll.dll-->NtOpenDirectoryObject Code Mismatch 0x7C90D550 + 4 [0C E9 06 2B B1 01]
ntdll.dll-->NtOpenEvent Inline - RelativeJump 0x7C90D560-->7D24CBA0 [SbieDll.dll]
ntdll.dll-->NtOpenEvent Inline - RelativeJump 0x7C90D565-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenEventPair Code Mismatch 0x7C90D570 + 4 [0C E9 E6 2A B1 01]
ntdll.dll-->NtOpenFile Inline - RelativeJump 0x7C90D580-->7D23EB90 [SbieDll.dll]
ntdll.dll-->NtOpenFile Inline - RelativeJump 0x7C90D585-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenIoCompletion Code Mismatch 0x7C90D590 + 4 [0C E9 C6 2A B1 01]
ntdll.dll-->NtOpenJobObject Code Mismatch 0x7C90D5A0 + 4 [0C E9 B6 2A B1 01]
ntdll.dll-->NtOpenKey Inline - RelativeJump 0x7C90D5B0-->7D252850 [SbieDll.dll]
ntdll.dll-->NtOpenKey Inline - RelativeJump 0x7C90D5B5-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenKeyedEvent Code Mismatch 0x7C90DFC0 + 4 [0C E9 96 20 B1 01]
ntdll.dll-->NtOpenMutant Inline - RelativeJump 0x7C90D5C0-->7D24CED0 [SbieDll.dll]
ntdll.dll-->NtOpenMutant Inline - RelativeJump 0x7C90D5C5-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenObjectAuditAlarm Code Mismatch 0x7C90D5D0 + 4 [30 E9 86 2A B1 01]
ntdll.dll-->NtOpenProcess Inline - RelativeJump 0x7C90D5E0-->7D25E990 [SbieDll.dll]
ntdll.dll-->NtOpenProcess Inline - RelativeJump 0x7C90D5E5-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenProcessToken Code Mismatch 0x7C90D5F0 + 4 [0C E9 66 2A B1 01]
ntdll.dll-->NtOpenProcessTokenEx Code Mismatch 0x7C90D600 + 4 [10 E9 56 2A B1 01]
ntdll.dll-->NtOpenSection Inline - RelativeJump 0x7C90D610-->7D24D5D0 [SbieDll.dll]
ntdll.dll-->NtOpenSection Inline - RelativeJump 0x7C90D615-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenSemaphore Inline - RelativeJump 0x7C90D620-->7D24D230 [SbieDll.dll]
ntdll.dll-->NtOpenSemaphore Inline - RelativeJump 0x7C90D625-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenSymbolicLinkObject Code Mismatch 0x7C90D630 + 4 [0C E9 26 2A B1 01]
ntdll.dll-->NtOpenThread Inline - RelativeJump 0x7C90D640-->7D25E020 [SbieDll.dll]
ntdll.dll-->NtOpenThread Inline - RelativeJump 0x7C90D645-->7E420060 [unknown_code_page]
ntdll.dll-->NtOpenThreadToken Code Mismatch 0x7C90D650 + 4 [10 E9 06 2A B1 01]
ntdll.dll-->NtOpenThreadTokenEx Code Mismatch 0x7C90D660 + 4 [14 E9 F6 29 B1 01]
ntdll.dll-->NtOpenTimer Code Mismatch 0x7C90D670 + 4 [0C E9 E6 29 B1 01]
ntdll.dll-->NtPlugPlayControl Code Mismatch 0x7C90D680 + 4 [0C E9 D6 29 B1 01]
ntdll.dll-->NtPowerInformation Code Mismatch 0x7C90D690 + 4 [14 E9 C6 29 B1 01]
ntdll.dll-->NtPrivilegeCheck Code Mismatch 0x7C90D6A0 + 4 [0C E9 B6 29 B1 01]
ntdll.dll-->NtPrivilegedServiceAuditAlarm Code Mismatch 0x7C90D6C0 + 4 [14 E9 96 29 B1 01]
ntdll.dll-->NtPrivilegeObjectAuditAlarm Code Mismatch 0x7C90D6B0 + 4 [18 E9 A6 29 B1 01]
ntdll.dll-->NtProtectVirtualMemory Code Mismatch 0x7C90D6D0 + 4 [14 E9 86 29 B1 01]
ntdll.dll-->NtPulseEvent Inline - RelativeJump 0x7C90D6E4-->7C90D715 [ntdll.dll]
ntdll.dll-->NtQueryAttributesFile Inline - RelativeJump 0x7C90D6F0-->7D23BF60 [SbieDll.dll]
ntdll.dll-->NtQueryAttributesFile Inline - RelativeJump 0x7C90D6F5-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryBootEntryOrder Code Mismatch 0x7C90D700 + 4 [08 E9 56 29 B1 01]
ntdll.dll-->NtQueryBootOptions Code Mismatch 0x7C90D710 + 4 [08 E9 46 29 B1 01]
```

```
ntdll.dll-->NtQueryDebugFilterState Code Mismatch 0x7C90D720 + 4 [08 E9 36 29 B1 01]
ntdll.dll-->NtQueryDefaultLocale Code Mismatch 0x7C90D730 + 4 [08 E9 26 29 B1 01]
ntdll.dll-->NtQueryDefaultUILanguage Code Mismatch 0x7C90D740 + 4 [04 E9 16 29 B1 01]
ntdll.dll-->NtQueryDirectoryFile Inline - RelativeJump 0x7C90D750-->7D23A990 [SbieDll.dll]
ntdll.dll-->NtQueryDirectoryFile Inline - RelativeJump 0x7C90D755-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryDirectoryObject Code Mismatch 0x7C90D760 + 4 [1C E9 F6 28 B1 01]
ntdll.dll-->NtQueryEaFile Code Mismatch 0x7C90D770 + 4 [24 E9 E6 28 B1 01]
ntdll.dll-->NtQueryEvent Code Mismatch 0x7C90D780 + 4 [14 E9 D6 28 B1 01]
ntdll.dll-->NtQueryFullAttributesFile Inline - RelativeJump 0x7C90D790-->7D23BFC0 [SbieDll.dll]
ntdll.dll-->NtQueryFullAttributesFile Inline - RelativeJump 0x7C90D795-->7E420060
[unknown_code_page]
ntdll.dll-->NtQueryInformationAtom Code Mismatch 0x7C90D7A0 + 4 [14 E9 B6 28 B1 01]
ntdll.dll-->NtQueryInformationFile Inline - RelativeJump 0x7C90D7B0-->7D23E450 [SbieDll.dll]
ntdll.dll-->NtQueryInformationFile Inline - RelativeJump 0x7C90D7B5-->7E420060
[unknown_code_page]
ntdll.dll-->NtQueryInformationJobObject Code Mismatch 0x7C90D7C0 + 4 [14 E9 96 28 B1 01]
ntdll.dll-->NtQueryInformationPort Code Mismatch 0x7C90D7D0 + 4 [14 E9 86 28 B1 01]
ntdll.dll-->NtQueryInformationProcess Inline - RelativeJump 0x7C90D7E4-->7C90D814 [ntdll.dll]
ntdll.dll-->NtQueryInformationThread Code Mismatch 0x7C90D7F0 + 4 [14 E9 66 28 B1 01]
ntdll.dll-->NtQueryInformationToken Code Mismatch 0x7C90D800 + 4 [14 E9 56 28 B1 01]
ntdll.dll-->NtQueryInstallUILanguage Code Mismatch 0x7C90D810 + 4 [04 E9 46 28 B1 01]
ntdll.dll-->NtQueryIntervalProfile Code Mismatch 0x7C90D820 + 4 [08 E9 36 28 B1 01]
ntdll.dll-->NtQueryIoCompletion Code Mismatch 0x7C90D830 + 4 [14 E9 26 28 B1 01]
ntdll.dll-->NtQueryKey Inline - RelativeJump 0x7C90D840-->7D252C70 [SbieDll.dll]
ntdll.dll-->NtQueryKey Inline - RelativeJump 0x7C90D845-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryMultipleValueKey Inline - RelativeJump 0x7C90D850-->7D252620 [SbieDll.dll]
ntdll.dll-->NtQueryMultipleValueKey Inline - RelativeJump 0x7C90D855-->7E420060
[unknown_code_page]
ntdll.dll-->NtQueryMutant Code Mismatch 0x7C90D860 + 4 [14 E9 F6 27 B1 01]
ntdll.dll-->NtQueryObject Inline - RelativeJump 0x7C90D870-->7D254F00 [SbieDll.dll]
ntdll.dll-->NtQueryObject Inline - RelativeJump 0x7C90D875-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryOpenSubKeys Code Mismatch 0x7C90D880 + 4 [08 E9 D6 27 B1 01]
ntdll.dll-->NtQueryPerformanceCounter Code Mismatch 0x7C90D890 + 4 [08 E9 C6 27 B1 01]
ntdll.dll-->NtQueryPortInformationProcess Inline - RelativeJump 0x7C90DFF5-->7E420060
[unknown_code_page]
ntdll.dll-->NtQueryQuotaInformationFile Code Mismatch 0x7C90D8A0 + 4 [24 E9 B6 27 B1 01]
ntdll.dll-->NtQuerySection Code Mismatch 0x7C90D8B0 + 4 [14 E9 A6 27 B1 01]
ntdll.dll-->NtQuerySecurityObject Inline - RelativeJump 0x7C90D8C0-->7D25E080 [SbieDll.dll]
ntdll.dll-->NtQuerySecurityObject Inline - RelativeJump 0x7C90D8C5-->7E420060 [unknown_code_page]
ntdll.dll-->NtQuerySemaphore Code Mismatch 0x7C90D8D0 + 4 [14 E9 86 27 B1 01]
ntdll.dll-->NtQuerySymbolicLinkObject Inline - RelativeJump 0x7C90D8E4-->7C90D913 [ntdll.dll]
ntdll.dll-->NtQuerySystemEnvironmentValue Code Mismatch 0x7C90D8F0 + 4 [10 E9 66 27 B1 01]
ntdll.dll-->NtQuerySystemEnvironmentValueEx Code Mismatch 0x7C90D900 + 4 [14 E9 56 27 B1 01]
ntdll.dll-->NtQuerySystemInformation Inline - RelativeJump 0x7C90D910-->7D260870 [SbieDll.dll]
ntdll.dll-->NtQuerySystemInformation Inline - RelativeJump 0x7C90D915-->7E420060
[unknown_code_page]
ntdll.dll-->NtQuerySystemTime Code Mismatch 0x7C90D920 + 4 [04 E9 36 27 B1 01]
ntdll.dll-->NtQueryTimer Code Mismatch 0x7C90D930 + 4 [14 E9 26 27 B1 01]
ntdll.dll-->NtQueryTimerResolution Code Mismatch 0x7C90D940 + 4 [0C E9 16 27 B1 01]
ntdll.dll-->NtQueryValueKey Inline - RelativeJump 0x7C90D950-->7D252210 [SbieDll.dll]
ntdll.dll-->NtQueryValueKey Inline - RelativeJump 0x7C90D955-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryVirtualMemory Inline - RelativeJump 0x7C90D960-->7D2550F0 [SbieDll.dll]
ntdll.dll-->NtQueryVirtualMemory Inline - RelativeJump 0x7C90D965-->7E420060 [unknown_code_page]
ntdll.dll-->NtQueryVolumeInformationFile Inline - RelativeJump 0x7C90D970-->7D23CE50
[SbieDll.dll]
ntdll.dll-->NtQueryVolumeInformationFile Inline - RelativeJump 0x7C90D975-->7E420060
[unknown_code_page]
ntdll.dll-->NtQueueApcThread Code Mismatch 0x7C90D980 + 4 [14 E9 D6 26 B1 01]
ntdll.dll-->NtRaiseHardError Code Mismatch 0x7C90D9A0 + 4 [18 E9 B6 26 B1 01]
ntdll.dll-->NtReadFile Inline - RelativeJump 0x7C90D9B0-->7D233290 [SbieDll.dll]
ntdll.dll-->NtReadFile Inline - RelativeJump 0x7C90D9B5-->7E420060 [unknown_code_page]
ntdll.dll-->NtReadFileScatter Code Mismatch 0x7C90D9C0 + 4 [24 E9 96 26 B1 01]
ntdll.dll-->NtReadRequestData Code Mismatch 0x7C90D9D0 + 4 [18 E9 86 26 B1 01]
ntdll.dll-->NtReadVirtualMemory Inline - RelativeJump 0x7C90D9E4-->7C90DA12 [ntdll.dll]
ntdll.dll-->NtRegisterThreadTerminatePort Code Mismatch 0x7C90D9F0 + 4 [04 E9 66 26 B1 01]
ntdll.dll-->NtReleaseKeyedEvent Code Mismatch 0x7C90DFD0 + 4 [10 E9 86 20 B1 01]
ntdll.dll-->NtReleaseMutant Code Mismatch 0x7C90DA00 + 4 [08 E9 56 26 B1 01]
ntdll.dll-->NtReleaseSemaphore Code Mismatch 0x7C90DA10 + 4 [0C E9 46 26 B1 01]
ntdll.dll-->NtRemoveIoCompletion Code Mismatch 0x7C90DA20 + 4 [14 E9 36 26 B1 01]
ntdll.dll-->NtRemoveProcessDebug Code Mismatch 0x7C90DA30 + 4 [08 E9 26 26 B1 01]
ntdll.dll-->NtRenameKey Inline - RelativeJump 0x7C90DA40-->7D250340 [SbieDll.dll]
ntdll.dll-->NtRenameKey Inline - RelativeJump 0x7C90DA45-->7E420060 [unknown_code_page]
ntdll.dll-->NtReplaceKey Code Mismatch 0x7C90DA50 + 4 [0C E9 06 26 B1 01]
ntdll.dll-->NtReplyPort Code Mismatch 0x7C90DA60 + 4 [08 E9 F6 25 B1 01]
ntdll.dll-->NtReplyWaitReceivePort Code Mismatch 0x7C90DA70 + 4 [10 E9 E6 25 B1 01]
ntdll.dll-->NtReplyWaitReceivePortEx Code Mismatch 0x7C90DA80 + 4 [14 E9 D6 25 B1 01]
```

```
ntdll.dll-->NtReplyWaitReplyPort Code Mismatch 0x7C90DA90 + 4 [08 E9 C6 25 B1 01]
ntdll.dll-->NtRequestDeviceWakeup Code Mismatch 0x7C90DAA0 + 4 [04 E9 B6 25 B1 01]
ntdll.dll-->NtRequestPort Code Mismatch 0x7C90DAB0 + 4 [08 E9 A6 25 B1 01]
ntdll.dll-->NtRequestWaitReplyPort Inline - RelativeJump 0x7C90DAC0-->7D24B160 [SbieDll.dll]
ntdll.dll-->NtRequestWaitReplyPort Inline - RelativeJump 0x7C90DAC5-->7E420060
[unknown_code_page]
ntdll.dll-->NtRequestWakeupLatency Code Mismatch 0x7C90DAD0 + 4 [04 E9 86 25 B1 01]
ntdll.dll-->NtResetEvent Inline - RelativeJump 0x7C90DAE4-->7C90DB11 [ntdll.dll]
ntdll.dll-->NtResetWriteWatch Code Mismatch 0x7C90DAF0 + 4 [0C E9 66 25 B1 01]
ntdll.dll-->NtRestoreKey Code Mismatch 0x7C90DB00 + 4 [0C E9 56 25 B1 01]
ntdll.dll-->NtResumeProcess Code Mismatch 0x7C90DB10 + 4 [04 E9 46 25 B1 01]
ntdll.dll-->NtResumeThread Code Mismatch 0x7C90DB20 + 4 [08 E9 36 25 B1 01]
ntdll.dll-->NtSaveKey Inline - RelativeJump 0x7C90DB30-->7D262800 [SbieDll.dll]
ntdll.dll-->NtSaveKey Inline - RelativeJump 0x7C90DB35-->7E420060 [unknown_code_page]
ntdll.dll-->NtSaveKeyEx Code Mismatch 0x7C90DB40 + 4 [0C E9 16 25 B1 01]
ntdll.dll-->NtSaveMergedKeys Code Mismatch 0x7C90DB50 + 4 [0C E9 06 25 B1 01]
ntdll.dll-->NtSecureConnectPort Inline - RelativeJump 0x7C90DB60-->7D24C090 [SbieDll.dll]
ntdll.dll-->NtSecureConnectPort Inline - RelativeJump 0x7C90DB65-->7E420060 [unknown_code_page]
ntdll.dll-->NtSetBootEntryOrder Code Mismatch 0x7C90DB70 + 4 [08 E9 E6 24 B1 01]
ntdll.dll-->NtSetBootOptions Code Mismatch 0x7C90DB80 + 4 [08 E9 D6 24 B1 01]
ntdll.dll-->NtSetContextThread Code Mismatch 0x7C90DB90 + 4 [08 E9 C6 24 B1 01]
ntdll.dll-->NtSetDebugFilterState Code Mismatch 0x7C90DBA0 + 4 [0C E9 B6 24 B1 01]
ntdll.dll-->NtSetDefaultHardErrorPort Code Mismatch 0x7C90DBB0 + 4 [04 E9 A6 24 B1 01]
ntdll.dll-->NtSetDefaultLocale Code Mismatch 0x7C90DBC0 + 4 [08 E9 96 24 B1 01]
ntdll.dll-->NtSetDefaultUILanguage Code Mismatch 0x7C90DBD0 + 4 [04 E9 86 24 B1 01]
ntdll.dll-->NtSetEaFile Inline - RelativeJump 0x7C90DBE4-->7C90DC10 [ntdll.dll]
ntdll.dll-->NtSetEvent Code Mismatch 0x7C90DBF0 + 4 [08 E9 66 24 B1 01]
ntdll.dll-->NtSetEventBoostPriority Code Mismatch 0x7C90DC00 + 4 [04 E9 56 24 B1 01]
ntdll.dll-->NtSetHighEventPair Code Mismatch 0x7C90DC10 + 4 [04 E9 46 24 B1 01]
ntdll.dll-->NtSetHighWaitLowEventPair Code Mismatch 0x7C90DC20 + 4 [04 E9 36 24 B1 01]
ntdll.dll-->NtSetInformationDebugObject Code Mismatch 0x7C90DC30 + 4 [14 E9 26 24 B1 01]
ntdll.dll-->NtSetInformationFile Inline - RelativeJump 0x7C90DC40-->7D23E850 [SbieDll.dll]
ntdll.dll-->NtSetInformationFile Inline - RelativeJump 0x7C90DC45-->7E420060 [unknown_code_page]
ntdll.dll-->NtSetInformationJobObject Inline - RelativeJump 0x7C90DC50-->7D260970 [SbieDll.dll]
ntdll.dll-->NtSetInformationJobObject Inline - RelativeJump 0x7C90DC55-->7E420060
[unknown_code_page]
ntdll.dll-->NtSetInformationKey Code Mismatch 0x7C90DC60 + 4 [10 E9 F6 23 B1 01]
ntdll.dll-->NtSetInformationObject Code Mismatch 0x7C90DC70 + 4 [10 E9 E6 23 B1 01]
ntdll.dll-->NtSetInformationProcess Inline - RelativeJump 0x7C90DC80-->7D256190 [SbieDll.dll]
ntdll.dll-->NtSetInformationProcess Inline - RelativeJump 0x7C90DC85-->7E420060
[unknown_code_page]
ntdll.dll-->NtSetInformationThread Code Mismatch 0x7C90DC90 + 4 [10 E9 C6 23 B1 01]
ntdll.dll-->NtSetInformationToken Inline - RelativeJump 0x7C90DCA0-->7D25E2B0 [SbieDll.dll]
ntdll.dll-->NtSetInformationToken Inline - RelativeJump 0x7C90DCA5-->7E420060 [unknown_code_page]
ntdll.dll-->NtSetIntervalProfile Code Mismatch 0x7C90DCB0 + 4 [08 E9 A6 23 B1 01]
ntdll.dll-->NtSetIoCompletion Code Mismatch 0x7C90DCC0 + 4 [14 E9 96 23 B1 01]
ntdll.dll-->NtSetLdtEntries Code Mismatch 0x7C90DCD0 + 4 [18 E9 86 23 B1 01]
ntdll.dll-->NtSetLowEventPair Inline - RelativeJump 0x7C90DCE4-->7C90DD0F [ntdll.dll]
ntdll.dll-->NtSetLowWaitHighEventPair Code Mismatch 0x7C90DCF0 + 4 [04 E9 66 23 B1 01]
ntdll.dll-->NtSetQuotaInformationFile Code Mismatch 0x7C90DD00 + 4 [10 E9 56 23 B1 01]
ntdll.dll-->NtSetSecurityObject Inline - RelativeJump 0x7C90DD10-->7D25E150 [SbieDll.dll]
ntdll.dll-->NtSetSecurityObject Inline - RelativeJump 0x7C90DD15-->7E420060 [unknown_code_page]
ntdll.dll-->NtSetSystemEnvironmentValue Code Mismatch 0x7C90DD20 + 4 [08 E9 36 23 B1 01]
ntdll.dll-->NtSetSystemEnvironmentValueEx Code Mismatch 0x7C90DD30 + 4 [14 E9 26 23 B1 01]
ntdll.dll-->NtSetSystemInformation Code Mismatch 0x7C90DD40 + 4 [0C E9 16 23 B1 01]
ntdll.dll-->NtSetSystemPowerState Code Mismatch 0x7C90DD50 + 4 [0C E9 06 23 B1 01]
ntdll.dll-->NtSetSystemTime Code Mismatch 0x7C90DD60 + 4 [08 E9 F6 22 B1 01]
ntdll.dll-->NtSetThreadExecutionState Code Mismatch 0x7C90DD70 + 4 [08 E9 E6 22 B1 01]
ntdll.dll-->NtSetTimer Code Mismatch 0x7C90DD80 + 4 [1C E9 D6 22 B1 01]
ntdll.dll-->NtSetTimerResolution Code Mismatch 0x7C90DD90 + 4 [0C E9 C6 22 B1 01]
ntdll.dll-->NtSetUuidSeed Code Mismatch 0x7C90DDA0 + 4 [04 E9 B6 22 B1 01]
ntdll.dll-->NtSetValueKey Inline - RelativeJump 0x7C90DDB0-->7D250C90 [SbieDll.dll]
ntdll.dll-->NtSetValueKey Inline - RelativeJump 0x7C90DDB5-->7E420060 [unknown_code_page]
ntdll.dll-->NtSetVolumeInformationFile Code Mismatch 0x7C90DDC0 + 4 [14 E9 96 22 B1 01]
ntdll.dll-->NtShutdownSystem Code Mismatch 0x7C90DDD0 + 4 [04 E9 86 22 B1 01]
ntdll.dll-->NtSignalAndWaitForSingleObject Inline - RelativeJump 0x7C90DDE4-->7C90DE0E
[ntdll.dll]
ntdll.dll-->NtStartProfile Code Mismatch 0x7C90DDF0 + 4 [04 E9 66 22 B1 01]
ntdll.dll-->NtStopProfile Code Mismatch 0x7C90DE00 + 4 [04 E9 56 22 B1 01]
ntdll.dll-->NtSuspendProcess Code Mismatch 0x7C90DE10 + 4 [04 E9 46 22 B1 01]
ntdll.dll-->NtSuspendThread Code Mismatch 0x7C90DE20 + 4 [08 E9 36 22 B1 01]
ntdll.dll-->NtSystemDebugControl Code Mismatch 0x7C90DE30 + 4 [18 E9 26 22 B1 01]
ntdll.dll-->NtTestAlert Inline - RelativeJump 0x7C90DE75-->7E420060 [unknown_code_page]
ntdll.dll-->NtTraceEvent Inline - RelativeJump 0x7C90DE80-->7D260640 [SbieDll.dll]
ntdll.dll-->NtTraceEvent Inline - RelativeJump 0x7C90DE85-->7E420060 [unknown_code_page]
ntdll.dll-->NtTranslateFilePath Code Mismatch 0x7C90DE90 + 4 [10 E9 C6 21 B1 01]
```

```
ntdll.dll-->NtUnloadDriver Code Mismatch 0x7C90DEA0 + 4 [04 E9 B6 21 B1 01]
ntdll.dll-->NtUnloadKey Code Mismatch 0x7C90DEB0 + 4 [04 E9 A6 21 B1 01]
ntdll.dll-->NtUnloadKeyEx Code Mismatch 0x7C90DEC0 + 4 [08 E9 96 21 B1 01]
ntdll.dll-->NtUnlockFile Code Mismatch 0x7C90DED0 + 4 [14 E9 86 21 B1 01]
ntdll.dll-->NtUnlockVirtualMemory Inline - RelativeJump 0x7C90DEE4-->7C90DF0D [ntdll.dll]
ntdll.dll-->NtUnmapViewOfSection Code Mismatch 0x7C90DEF0 + 4 [08 E9 66 21 B1 01]
ntdll.dll-->NtVdmControl Code Mismatch 0x7C90DF00 + 4 [08 E9 56 21 B1 01]
ntdll.dll-->NtWaitForDebugEvent Code Mismatch 0x7C90DF10 + 4 [10 E9 46 21 B1 01]
ntdll.dll-->NtWaitForKeyedEvent Inline - RelativeJump 0x7C90DFE4-->7C90E00C [ntdll.dll]
ntdll.dll-->NtWaitForMultipleObjects Code Mismatch 0x7C90DF20 + 4 [14 E9 36 21 B1 01]
ntdll.dll-->NtWaitForSingleObject Code Mismatch 0x7C90DF30 + 4 [0C E9 26 21 B1 01]
ntdll.dll-->NtWaitHighEventPair Code Mismatch 0x7C90DF40 + 4 [04 E9 16 21 B1 01]
ntdll.dll-->NtWaitLowEventPair Code Mismatch 0x7C90DF50 + 4 [04 E9 06 21 B1 01]
ntdll.dll-->NtWriteFile Inline - RelativeJump 0x7C90DF60-->7D233360 [SbieDll.dll]
ntdll.dll-->NtWriteFile Inline - RelativeJump 0x7C90DF65-->7E420060 [unknown_code_page]
ntdll.dll-->NtWriteFileGather Code Mismatch 0x7C90DF70 + 4 [24 E9 E6 20 B1 01]
ntdll.dll-->NtWriteRequestData Code Mismatch 0x7C90DF80 + 4 [18 E9 D6 20 B1 01]
ntdll.dll-->NtWriteVirtualMemory Code Mismatch 0x7C90DF90 + 4 [14 E9 C6 20 B1 01]
ntdll.dll-->NtYieldExecution Inline - RelativeJump 0x7C90DFA5-->7E420060 [unknown_code_page]
ntdll.dll-->RtlGetCurrentDirectory_U Inline - RelativeJump 0x7C9144E6-->7D23B120 [SbieDll.dll]
ntdll.dll-->RtlGetFullPathName_U Inline - RelativeJump 0x7C914389-->7D23B3E0 [SbieDll.dll]
ntdll.dll-->RtlSetCurrentDirectory_U Inline - RelativeJump 0x7C91E78E-->7D23B360 [SbieDll.dll]
shell32.dll-->ShellExecuteExW Inline - RelativeJump 0x7CA02F03-->7D25F250 [SbieDll.dll]
shell32.dll-->SHOpenFolderAndSelectItems Inline - RelativeJump 0x7CAC2A1E-->7D2601C0
[SbieDll.dll]
user32.dll+0x00002998 Inline - RelativeJump 0x7E362998-->7E3629EE [user32.dll]
user32.dll+0x000086DB Inline - RelativeJump 0x7E3686DB-->7D244690 [SbieDll.dll]
user32.dll-->ActivateKeyboardLayout Inline - RelativeJump 0x7E378673-->7D240400 [SbieDll.dll]
user32.dll-->AnimateWindow Inline - RelativeJump 0x7E372156-->7D2405A0 [SbieDll.dll]
user32.dll-->ClientToScreen Inline - RelativeJump 0x7E379B60-->7D240F60 [SbieDll.dll]
user32.dll-->ClipCursor Inline - RelativeJump 0x7E38FDC5-->7D246D90 [SbieDll.dll]
user32.dll-->CloseClipboard Inline - RelativeJump 0x7E380265-->7D247260 [SbieDll.dll]
user32.dll-->CreateDesktopA Inline - RelativeJump 0x7E3A5E37-->7D244900 [SbieDll.dll]
user32.dll-->CreateDesktopW Inline - RelativeJump 0x7E37162A-->7D244900 [SbieDll.dll]
user32.dll-->CreateDialogIndirectParamA Inline - RelativeJump 0x7E389B28-->7D248EB0 [SbieDll.dll]
user32.dll-->CreateDialogIndirectParamAorW Inline - RelativeJump 0x7E37680B-->7D248DD0
[SbieDll.dll]
user32.dll-->CreateDialogIndirectParamW Inline - RelativeJump 0x7E38F01F-->7D248E80 [SbieDll.dll]
user32.dll-->CreateDialogParamA Inline - RelativeJump 0x7E38C7DB-->7D248F80 [SbieDll.dll]
user32.dll-->CreateDialogParamW Inline - RelativeJump 0x7E36EA3B-->7D248F40 [SbieDll.dll]
user32.dll-->CreateWindowExA Inline - RelativeJump 0x7E37E4A9-->7D23FFF0 [SbieDll.dll]
user32.dll-->CreateWindowExW Inline - RelativeJump 0x7E37D0A3-->7D23FE50 [SbieDll.dll]
user32.dll-->DdeInitializeA Inline - RelativeJump 0x7E3AA8F6-->7D243E60 [SbieDll.dll]
user32.dll-->DdeInitializeW Inline - RelativeJump 0x7E3706D7-->7D243E10 [SbieDll.dll]
user32.dll-->DefWindowProcA Inline - RelativeJump 0x7E37C17E-->7D240390 [SbieDll.dll]
user32.dll-->DefWindowProcW Inline - RelativeJump 0x7E378D20-->7D240320 [SbieDll.dll]
user32.dll-->DialogBoxIndirectParamA Inline - RelativeJump 0x7E3A6D7D-->7D248F10 [SbieDll.dll]
user32.dll-->DialogBoxIndirectParamAorW Inline - RelativeJump 0x7E3749D0-->7D248E30 [SbieDll.dll]
user32.dll-->DialogBoxIndirectParamW Inline - RelativeJump 0x7E382072-->7D248EE0 [SbieDll.dll]
user32.dll-->DialogBoxParamA Inline - RelativeJump 0x7E38B144-->7D249000 [SbieDll.dll]
user32.dll-->DialogBoxParamW Inline - RelativeJump 0x7E3747AB-->7D248FC0 [SbieDll.dll]
user32.dll-->DispatchMessageA Inline - RelativeJump 0x7E3696B8-->7D246120 [SbieDll.dll]
user32.dll-->DispatchMessageW Inline - RelativeJump 0x7E368A01-->7D246140 [SbieDll.dll]
user32.dll-->EndTask Inline - RelativeJump 0x7E3AA0A5-->7D2404D0 [SbieDll.dll]
user32.dll-->EnumChildWindows Inline - RelativeJump 0x7E37B0F0-->7D2447F0 [SbieDll.dll]
user32.dll-->EnumDesktopsA Inline - RelativeJump 0x7E38234B-->7D2448C0 [SbieDll.dll]
user32.dll-->EnumDesktopsW Inline - RelativeJump 0x7E37853B-->7D2448A0 [SbieDll.dll]
user32.dll-->EnumDesktopWindows Inline - RelativeJump 0x7E37851A-->7D244880 [SbieDll.dll]
user32.dll-->EnumThreadWindows Inline - RelativeJump 0x7E37F539-->7D244840 [SbieDll.dll]
user32.dll-->EnumWindows Inline - RelativeJump 0x7E37A5AE-->7D2447D0 [SbieDll.dll]
user32.dll-->ExitWindowsEx Inline - RelativeJump 0x7E3AA275-->7D240430 [SbieDll.dll]
user32.dll-->FindWindowA Inline - RelativeJump 0x7E3782E1-->7D244D50 [SbieDll.dll]
user32.dll-->FindWindowExA Inline - RelativeJump 0x7E38214A-->7D244EC0 [SbieDll.dll]
user32.dll-->FindWindowExW Inline - RelativeJump 0x7E36E0E3-->7D244E00 [SbieDll.dll]
user32.dll-->FindWindowW Inline - RelativeJump 0x7E37C9C3-->7D244CA0 [SbieDll.dll]
user32.dll-->GetClassInfoA Inline - RelativeJump 0x7E38EBFF-->7D242FF0 [SbieDll.dll]
user32.dll-->GetClassInfoExA Inline - RelativeJump 0x7E36DD58-->7D242F10 [SbieDll.dll]
user32.dll-->GetClassInfoExW Inline - RelativeJump 0x7E36DEBC-->7D242EA0 [SbieDll.dll]
user32.dll-->GetClassInfoW Inline - RelativeJump 0x7E37E81E-->7D242F80 [SbieDll.dll]
user32.dll-->GetClassLongA Inline - RelativeJump 0x7E37F4F1-->7D245A90 [SbieDll.dll]
user32.dll-->GetClassLongW Inline - RelativeJump 0x7E379AE9-->7D245A70 [SbieDll.dll]
user32.dll-->GetClassNameA Inline - RelativeJump 0x7E37F45F-->7D243190 [SbieDll.dll]
user32.dll-->GetClassNameW Inline - RelativeJump 0x7E379D12-->7D243060 [SbieDll.dll]
user32.dll-->GetClientRect Inline - RelativeJump 0x7E37908E-->7D241060 [SbieDll.dll]
user32.dll-->GetClipboardData Inline - RelativeJump 0x7E380DBA-->7D247790 [SbieDll.dll]
user32.dll-->GetIconInfo Inline - RelativeJump 0x7E37D427-->7D246F00 [SbieDll.dll]
```

```
user32.dll-->GetParent Inline - RelativeJump 0x7E37910F-->7D246CE0 [SbieDll.dll]
user32.dll-->GetPropA Inline - RelativeJump 0x7E380042-->7D2455F0 [SbieDll.dll]
user32.dll-->GetPropW Inline - RelativeJump 0x7E3794B3-->7D245550 [SbieDll.dll]
user32.dll-->GetShellWindow Inline - RelativeJump 0x7E369252-->7D244F90 [SbieDll.dll]
user32.dll-->GetUserObjectInformationW Inline - RelativeJump 0x7E368D17-->7D247190 [SbieDll.dll]
user32.dll-->GetWindow Inline - RelativeJump 0x7E379655-->7D246CB0 [SbieDll.dll]
user32.dll-->GetWindowInfo Inline - RelativeJump 0x7E37C49C-->7D2411A0 [SbieDll.dll]
user32.dll-->GetWindowLongA Inline - RelativeJump 0x7E36945D-->7D2459E0 [SbieDll.dll]
user32.dll-->GetWindowLongW Inline - RelativeJump 0x7E3688A6-->7D245950 [SbieDll.dll]
user32.dll-->GetWindowRect Inline - RelativeJump 0x7E3790B4-->7D241100 [SbieDll.dll]
user32.dll-->GetWindowTextA Inline - RelativeJump 0x7E38216B-->7D244500 [SbieDll.dll]
user32.dll-->GetWindowTextW Inline - RelativeJump 0x7E37A5CD-->7D2444D0 [SbieDll.dll]
user32.dll-->IsIconic Inline - RelativeJump 0x7E3797FF-->7D240E80 [SbieDll.dll]
user32.dll-->IsWindow Inline - RelativeJump 0x7E379313-->7D240E00 [SbieDll.dll]
user32.dll-->IsWindowEnabled Inline - RelativeJump 0x7E37977A-->7D240E20 [SbieDll.dll]
user32.dll-->IsWindowUnicode Inline - RelativeJump 0x7E379F72-->7D240E60 [SbieDll.dll]
user32.dll-->IsWindowVisible Inline - RelativeJump 0x7E379E3D-->7D240E40 [SbieDll.dll]
user32.dll-->IsZoomed Inline - RelativeJump 0x7E379C8A-->7D240EA0 [SbieDll.dll]
user32.dll-->MapWindowPoints Inline - RelativeJump 0x7E379507-->7D240EC0 [SbieDll.dll]
user32.dll-->MonitorFromWindow Inline - RelativeJump 0x7E37A679-->7D247050 [SbieDll.dll]
user32.dll-->MoveWindow Inline - RelativeJump 0x7E37B29E-->7D240480 [SbieDll.dll]
user32.dll-->OpenClipboard Inline - RelativeJump 0x7E380277-->7D247220 [SbieDll.dll]
user32.dll-->OpenDesktopA Inline - RelativeJump 0x7E382369-->7D2448E0 [SbieDll.dll]
user32.dll-->OpenDesktopW Inline - RelativeJump 0x7E378559-->7D2448E0 [SbieDll.dll]
user32.dll-->OpenInputDesktop Inline - RelativeJump 0x7E36ECA3-->7D247170 [SbieDll.dll]
user32.dll-->PostMessageA Inline - RelativeJump 0x7E37AAFD-->7D2469B0 [SbieDll.dll]
user32.dll-->PostMessageW Inline - RelativeJump 0x7E368CCB-->7D246A30 [SbieDll.dll]
user32.dll-->RegisterClassA Inline - RelativeJump 0x7E37EA5E-->7D242D10 [SbieDll.dll]
user32.dll-->RegisterClassExA Inline - RelativeJump 0x7E377C39-->7D242B70 [SbieDll.dll]
user32.dll-->RegisterClassExW Inline - RelativeJump 0x7E36AF7F-->7D242AA0 [SbieDll.dll]
user32.dll-->RegisterClassW Inline - RelativeJump 0x7E36A39A-->7D242C40 [SbieDll.dll]
user32.dll-->RegisterDeviceNotificationA Inline - RelativeJump 0x7E371B3B-->7D240560
[SbieDll.dll]
user32.dll-->RegisterDeviceNotificationW Inline - RelativeJump 0x7E36E8B9-->7D240560
[SbieDll.dll]
user32.dll-->RemovePropA Inline - RelativeJump 0x7E380094-->7D245780 [SbieDll.dll]
user32.dll-->RemovePropW Inline - RelativeJump 0x7E37C076-->7D245730 [SbieDll.dll]
user32.dll-->ScreenToClient Inline - RelativeJump 0x7E3797A0-->7D240FE0 [SbieDll.dll]
user32.dll-->SendMessageA Inline - RelativeJump 0x7E37F3C2-->7D246720 [SbieDll.dll]
user32.dll-->SendMessageTimeoutA Inline - RelativeJump 0x7E37FB6B-->7D246880 [SbieDll.dll]
user32.dll-->SendMessageTimeoutW Inline - RelativeJump 0x7E37CDAA-->7D2468C0 [SbieDll.dll]
user32.dll-->SendMessageW Inline - RelativeJump 0x7E37929A-->7D2467C0 [SbieDll.dll]
user32.dll-->SendNotifyMessageA Inline - RelativeJump 0x7E3A3948-->7D246950 [SbieDll.dll]
user32.dll-->SendNotifyMessageW Inline - RelativeJump 0x7E37D64F-->7D246980 [SbieDll.dll]
user32.dll-->SetCursor Inline - RelativeJump 0x7E379930-->7D246EB0 [SbieDll.dll]
user32.dll-->SetCursorPos Inline - RelativeJump 0x7E3A61B3-->7D246F50 [SbieDll.dll]
user32.dll-->SetDoubleClickTime Inline - RelativeJump 0x7E3A61CE-->7D246E40 [SbieDll.dll]
user32.dll-->SetForegroundWindow Inline - RelativeJump 0x7E3742ED-->7D246FD0 [SbieDll.dll]
user32.dll-->SetParent Inline - RelativeJump 0x7E37C7F9-->7D246D20 [SbieDll.dll]
user32.dll-->SetPropA Inline - RelativeJump 0x7E380000-->7D2456E0 [SbieDll.dll]
user32.dll-->SetPropW Inline - RelativeJump 0x7E37C0B9-->7D245690 [SbieDll.dll]
user32.dll-->SetThreadDesktop Inline - RelativeJump 0x7E377D2C-->7D23FC30 [SbieDll.dll]
user32.dll-->SetWindowLongA Inline - RelativeJump 0x7E37C29D-->7D245C20 [SbieDll.dll]
user32.dll-->SetWindowLongW Inline - RelativeJump 0x7E37C2BB-->7D245B90 [SbieDll.dll]
user32.dll-->SetWindowPos Inline - RelativeJump 0x7E3799F3-->7D240B80 [SbieDll.dll]
user32.dll-->SetWindowsHookExA Inline - RelativeJump 0x7E381211-->7D248260 [SbieDll.dll]
user32.dll-->SetWindowsHookExW Inline - RelativeJump 0x7E37820F-->7D2482C0 [SbieDll.dll]
user32.dll-->SwapMouseButton Inline - RelativeJump 0x7E3A6201-->7D246E40 [SbieDll.dll]
user32.dll-->SwitchDesktop Inline - RelativeJump 0x7E36FE6E-->7D23FC30 [SbieDll.dll]
user32.dll-->UnhookWindowsHookEx Inline - RelativeJump 0x7E37D5F3-->7D247F60 [SbieDll.dll]
user32.dll-->UnregisterClassA Inline - RelativeJump 0x7E3789A3-->7D242E40 [SbieDll.dll]
user32.dll-->UnregisterClassW Inline - RelativeJump 0x7E369AA4-->7D242DE0 [SbieDll.dll]
user32.dll-->UnregisterDeviceNotification Inline - RelativeJump 0x7E36E8D7-->7D23FC30
[SbieDll.dll]
user32.dll-->UserHandleGrantAccess Inline - RelativeJump 0x7E3BCFDA-->7D264CE0 [SbieDll.dll]
user32.dll-->WaitForInputIdle Inline - RelativeJump 0x7E39FAF5-->7D240620 [SbieDll.dll]
```

Due to lazy implementation, SBIEDLL does multiple improper modifications to the API call results, which however does not affect their work (not always, about a year ago author fixed one bug related to improper data modification – improper truncating result of some API call), but can be used as Sandboxie detection flags. For example, SBIEDLL truncates result of registry querying to hide from sandboxed application registry virtualization fact. Ironically, exactly this will be used to detect virtualization, as shown on Figure 3-16.

```
BOOL IsSandboxieVirtualRegistryPresent(
    VOID
    )
{
    BYTE IsSB = 0;
    ULONG hashA, hashB;
    HANDLE hKey;
    NTSTATUS Status;
    UNICODE_STRING ustrRegPath;
    OBJECT_ATTRIBUTES obja;

    WCHAR szObjectName[MAX_PATH * 2] = {0};

    hashA = HashFromStrW(REGSTR_KEY_USER);

    RtlInitUnicodeString(&ustrRegPath, REGSTR_KEY_USER);
    InitializeObjectAttributes(&obja, &ustrRegPath, OBJ_CASE_INSENSITIVE, NULL, NULL);
    Status = NtOpenKey(&hKey, MAXIMUM_ALLOWED, &obja);
    if (NT_SUCCESS(Status)) {
        if ( QueryObjectName((HKEY)hKey, &szObjectName, MAX_PATH * 2, TRUE) ) {
            hashB = HashFromStrW(szObjectName);
            if (hashB != hashA) IsSB = 1;
        }
        NtClose(hKey);
    }
    return IsSB;
}
```

FIGURE 3-16. VMDE Virtualization registry detection.

The following code used to detect if caller application is sandboxed, but only after caller make sure Sandboxie is present, so this code must work in complex with other detection methods to eliminate possible false positives. See Appendix for example of usage. Next, we will abuse helper-to-the-driver communication mechanism and use it to detect if the caller process is being sandboxed.

Another method based on how SBIEDLL communicates with Sandboxie driver that performs kernel mode modifications (and until recent versions driver was incompatible with Kernel Patch Protection due to obvious reasons). SBIEDLL does this through specially created device object. Internally SBIEDLL opens handle to it in every sandboxed process and uses it during *NtDeviceIoControlFile* calls, passing various IOCTL codes to the main driver. This device name is constant value:

SandboxieDriverApi

Following device name can be used to detect Sandboxie presence on a target machine (see part 2.1.2 of this document), because this device is permanent. We will enumerate caller process handle table and verify objects to belong to Sandboxie. To improve performance the verifying objects will be filtered by file object type. See Figure 3-17 for implementation details.

```c
        /* find sandboxie api device inside our handle table */
        hDummy = OpenDevice(L"\\Device\\Null", GENERIC_READ, &Status);
        if ( hDummy == NULL ) __leave;

        HandleTable = (PSYSTEM_HANDLE_INFORMATION)NativeAllocateInfoBuffer(SystemHandleInformation, &uLength);
        if ( HandleTable == NULL ) __leave;

        for (k=0; k<2; k++) {
            for (i=0; i<HandleTable->NumberOfHandles; i++) {
                if ( HandleTable->Handles[i].UniqueProcessId == OurID )
                    if (k == 0) {
                        if ( HandleTable->Handles[i].HandleValue == (USHORT)hDummy ) {
                            FileID = HandleTable->Handles[i].ObjectTypeIndex;
                            break;
                        }
                    } else {
                        if (HandleTable->Handles[i].ObjectTypeIndex == FileID )
                            if ( QueryObjectName((HANDLE)HandleTable->Handles[i].HandleValue, &szObjectName, MAX_PATH * 2, TRUE) ) {
                                if ( strstrW(szObjectName, VENDOR_SANDBOXIE) != NULL) {

#ifdef _DEBUG
                                    DebugLog(TEXT("Sandboxie::HandleTable"));
#endif

                                    IsSB = 1;
                                    break;
                                }
                            }
                    }
            }
        }
```

FIGURE 3-17. Enumerating handle table and identifying Sandboxie API device object.

Another way is to perform process memory scanning and searching for Sandboxie injected executable memory. How we can distinguish it from other kind of caller executable code? By tag Sandboxie uses for allocated memory. Yes, it is author nickname.

```
Command - Local kernel - WinDbg:6.11.0001.404 AM
lkd> .process /p fffffa80031ca5c0
Implicit process is now fffffa80`031ca5c0
lkd> db 000007F77E140000 l200
000007f7`7e140000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140010  00 00 00 00 00 00 00 00-00 01 14 7e f7 07 00 00  ...........~....
000007f7`7e140020  74 7a 75 6b ea 01 00 00-00 00 00 00 00 00 00 00  tzuk............
000007f7`7e140030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140080  ff ff 3f 00 00 00 00 00-00 00 00 00 00 00 00 00  ..?.............
000007f7`7e140090  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e1400a0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e1400b0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 c0  ................
000007f7`7e1400c0  ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff  ................
000007f7`7e1400d0  ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff  ................
000007f7`7e1400e0  ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff  ................
000007f7`7e1400f0  ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff  ................
000007f7`7e140100  74 7a 75 6b ff ff ff ff-ff ff ff ff ff ff ff ff  tzuk............
000007f7`7e140110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 14 7e  ...............~
000007f7`7e140130  f7 07 00 00 00 00 14 7e-f7 07 00 00 01 00 00 00  .......~........
000007f7`7e140140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000007f7`7e140190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

FIGURE 3-18. Sandboxie memory tag.

Now scan process memory, filter regions by allocation protect and inspect suspicious memory areas for known signature.

```
        /* brute-force memory to locate sandboxie injected code and locate sandboxie tag */
        if ( IsSB == 0 ) {
            i = (ULONG_PTR)g_siSysInfo.lpMinimumApplicationAddress;
            do {
                Status = NtQueryVirtualMemory(NtCurrentProcess(), (PVOID)i, MemoryBasicInformation,
                    &RegionInfo, sizeof(MEMORY_BASIC_INFORMATION), &uLength);
                if (NT_SUCCESS(Status)) {
                    if (IsExecutableCode(RegionInfo.AllocationProtect, RegionInfo.State)) {
                        for (k=i; k<i+RegionInfo.RegionSize; k+=sizeof(DWORD)) {
                            if (*(PDWORD)k == 'kuzt') {
                                IsSB = 1;

#ifdef _DEBUG
                                DebugLog(TEXT("Sandboxie::MemoryTag"));
#endif

                                break;
                            }
                        }
                    }
                    i += RegionInfo.RegionSize;
                } else {
                    i += PAGE_SIZE;
                }
            } while (i<(ULONG_PTR)g_siSysInfo.lpMaximumApplicationAddress);
        }
```

FIGURE 3-19. Looking for Sandboxie memory tag.

All the above detection methods must work only if Sandboxie presence confirmed by other detection methods and must work together to back up each other.

As a short conclusion: Sandboxie does so many modifications to the sandboxed process so detecting fact of sandboxing is a trivial task even if the above described methods will be out-dated.

2.4. Conclusion.

Malware more often use various VM detection tricks, targeting popular virtualization software. Currently these detection methods are primitive and blindly copy-pasted over various malware families. The virtualization software needs additional configuration to able to handle such cases. Unfortunately, none of the observed products has this feature out of the box. From all the listed products only VirtualBox and VMware has ability to additional reconfiguration for bypassing VM checks. As for Sandboxie, with its current implementation it cannot be hidden at all and not recommended as platform for malware research. More to say we would not recommend using this product on real PC on the regular basis as well, and if you still plan use it for malware research then do this only inside additionally controlled environment such as, you might already guess, virtual machine.

3. Appendix

3.1. Virtual PC detection

```
BOOL IsVirtualPC(
        VOID
        )
{
        BYTE IsVM = 0;
        ULONG dwDataSize = 0L;
        PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;

        /* devs of XP Mode we're so kind so they added special mutex, check it */
        IsVM = IsMutexExist(MUTEX_VPCXPMODE);
#ifdef _DEBUG
        if (IsVM == 1) DebugLog(TEXT("VPC::XPMode"));
#endif
        /*
           use well-known trick with illegal instructions, but be creative and don't use the
           same as here, there are numbers of them actually (> 1000), not only one set
        */
        if ( IsVM == 0 ) {
                __try {
                        __asm push   ebx
                        __asm mov    ebx, 0
                        __asm mov    eax, 1
                        __asm __emit 0Fh
                        __asm __emit 3Fh
                        __asm __emit 0Dh
                        __asm __emit 0h
                        __asm test   ebx, ebx
                        __asm setz   [IsVM]
                        __asm pop    ebx
                }
                __except(VPCExceptionHandler(GetExceptionInformation())) {  }
        }

#ifdef _DEBUG
        if (IsVM == 1) DebugLog(TEXT("VPC::Backdoor"));
#endif

        /* query virtual pc device */
        if ( IsVM == 0 ) {
                IsVM = (IsObjectExists(DEVICELINK, DEVICE_VIRTUALPC));

#ifdef _DEBUG
                if (IsVM == 1) DebugLog(TEXT("VPC::VMSvc"));
#endif

        }
        /* query virtual pc driver, reg. rights elevation */
        if ( IsVM == 0 ) {
                IsVM = (IsObjectExists(DRIVERLINK, DRIVER_VIRTUALPC));

#ifdef _DEBUG
                if (IsVM == 1) DebugLog(TEXT("VPC::VmDriver"));
#endif

        }

        /* scan raw firmware for specific string patterns */
        if (IsVM == 0) {
                pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, FIRM,
0xC0000);
                if (pSIF != NULL && dwDataSize > 0) {
                        IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_VPC, _strlenA(VENDOR_VPC));

#ifdef _DEBUG
                        if (IsVM == 1) DebugLog(TEXT("VPC::FirmwareS3MS"));
#endif
```

```
				RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
			}
		}

		/* scan raw smbios data for specific string patters */
		if ( IsVM == 0 ) {
			pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, RSMB, 0);
			if (pSIF != NULL && dwDataSize > 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, SMB_VPC, _strlenA(SMB_VPC));

#ifdef _DEBUG
				if (IsVM == 1) DebugLog(TEXT("VPC::VM"));
#endif

				RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
			}
		}

		/* query S3 VID on PCI bus devices */
		if ( IsVM == 0 ) {
			if ( vIsInList(VID_S3MS) != NULL ) IsVM = 1;
#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("VPC::VID_S3"));
#endif

		}
		return IsVM;
}
```

3.2. VMware detection

```
BOOL IsVMware(
	VOID
	)
{
	BYTE IsVM = 0;
	ULONG dwDataSize = 0L;
	PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;

	/* query VMware additions device presence */
	IsVM = IsObjectExists(DEVICELINK, DEVICE_VMWARE);

#ifdef _DEBUG
	if (IsVM == 1) DebugLog(TEXT("VMware::Memctrl"));
#endif

	if ( IsVM == 0 ) {
		/*
		   query VMware presence by hypervisor port

http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=100
9458
		*/
		__try {
			__asm push   edx
			__asm push   ecx
			__asm push   ebx
			__asm mov    eax, 'VMXh'
			__asm mov    ebx, 0
			__asm mov    ecx, 10
			__asm mov    edx, 'VX'
			__asm in     eax, dx
			__asm cmp    ebx, 'VMXh'
			__asm setz   [IsVM]
			__asm pop    ebx
			__asm pop    ecx
			__asm pop    edx
		}
		__except(EXCEPTION_EXECUTE_HANDLER)
		{
			IsVM = 0;
		}
	}

#ifdef _DEBUG
	if (IsVM == 1) DebugLog(TEXT("VMware::HVPort"));
#endif

	/* scan raw firmware for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, FIRM,
0xC0000);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_VMWARE,
_strlenA(VENDOR_VMWARE));

#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("VMware::Firmware_VMware"));
#endif

			RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
		}
	}

	/* scan raw SMBIOS firmware table for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, RSMB, 0);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_VMWARE,
_strlenA(VENDOR_VMWARE));
			if (IsVM == 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, SMB_VMWARE,
_strlenA(SMB_VMWARE));
#ifdef _DEBUG
```

```
                                if (IsVM == 1) DebugLog(TEXT("VMware::SMBIOS_VMware"));
#endif
                        }
                        RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                }
        }

        /* query VMware VID on PCI bus devices */
        if ( IsVM == 0 ) {
                if ( vIsInList(VID_VMWARE) != NULL ) IsVM = 1;

#ifdef _DEBUG
                if (IsVM == 1) DebugLog(TEXT("VMware::VID_VMware"));
#endif

        }

        return IsVM;
}
```

3.3. VirtualBox detection

```
BOOL IsVirtualBox(
	VOID
	)
{
	BYTE IsVM = 0;
	ULONG dwDataSize = 0L;
	PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;


	/* query vbox additions guest device */
	IsVM = (IsObjectExists(DEVICELINK, DEVICE_VIRTUALBOX1));

#ifdef _DEBUG
	if (IsVM == 1) DebugLog(TEXT("VBox::VBoxGuest"));
#endif

	/* query vbox additions device */
	if (IsVM == 0) {
		IsVM = (IsObjectExists(DEVICELINK, DEVICE_VIRTUALBOX2));

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("VBox::VBoxMiniRdrDN"));
#endif

	}
	/* query vbox additions video driver, reg. rights elevation*/
	if (IsVM == 0) {
		IsVM = (IsObjectExists(DRIVERLINK, DRIVER_VIRTUALBOX1));

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("VBox::VBoxVideo"));
#endif

	}
	/* query vbox additions mouse driver, reg. admin rights elevation */
	if (IsVM == 0) {
		IsVM = (IsObjectExists(DRIVERLINK, DRIVER_VIRTUALBOX2));

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("VBox::VBoxMouse"));
#endif

	}

	/* scan raw firmware for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, FIRM,
0xC0000);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_VBOX,
_strlenA(VENDOR_VBOX));

#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("VBox::FirmwareVBox"));
#endif

			if (IsVM == 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_ORACLE,
_strlenA(VENDOR_ORACLE));

#ifdef _DEBUG
				if (IsVM == 1) DebugLog(TEXT("VBox::FirmwareOracle"));
#endif

			}
			if (IsVM == 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_INNOTEK,
_strlenA(VENDOR_INNOTEK));

#ifdef _DEBUG
				if (IsVM == 1) DebugLog(TEXT("VBox::FirmwareInnotek"));
#endif
```

```
			}

			RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
		}
	}

	/* scan raw SMBIOS firmware table for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, RSMB, 0);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_VBOX,
_strlenA(VENDOR_VBOX));

#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("VBox::SMBIOS_VBox"));
#endif

			if (IsVM == 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_ORACLE,
_strlenA(VENDOR_ORACLE));
#ifdef _DEBUG
				if (IsVM == 1) DebugLog(TEXT("VBox::SMBIOS_Oracle"));
#endif

			}
			if (IsVM == 0) {
				IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_INNOTEK,
_strlenA(VENDOR_INNOTEK));
#ifdef _DEBUG
				if (IsVM == 1) DebugLog(TEXT("VBox::SMBIOS_Innotek"));
#endif
			}
			RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
		}
	}

	/* query oracle VID on PCI bus devices */
	if ( IsVM == 0 ) {
		if ( vIsInList(VID_ORACLE) != NULL ) IsVM = 1;

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("VBox::VID_Oracle"));
#endif

	}

	return IsVM;
}
```

3.4. Parallels detection

```
BOOL IsParallels(
	VOID
	)
{
	BYTE IsVM = 0;
	ULONG dwDataSize = 0L;
	PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;

	/* query parallels additions device presence */
	IsVM = IsObjectExists(DEVICELINK, DEVICE_PARALLELS1);

#ifdef _DEBUG
	if (IsVM == 1) DebugLog(TEXT("Parallels::prl_pv"));
#endif

	if ( IsVM == 0 ) {
		IsVM = IsObjectExists(DEVICELINK, DEVICE_PARALLELS2);

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("Parallels::prl_tg"));
#endif

	}
	if ( IsVM == 0 ) {
		IsVM = IsObjectExists(DEVICELINK, DEVICE_PARALLELS3);

#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("Parallels::prl_time"));
#endif

	}

	/* scan raw firmware for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, FIRM,
0xC0000);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, VENDOR_PARALLELS,
_strlenA(VENDOR_PARALLELS));

#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("Parallels::Firmware_Parallels"));
#endif


			RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
		}
	}

	/* scan raw SMBIOS firmware table for specific string patterns */
	if (IsVM == 0) {
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)GetFirmwareTable(&dwDataSize, RSMB, 0);
		if (pSIF != NULL && dwDataSize > 0) {
			IsVM = ScanDump((CHAR*)pSIF, dwDataSize, SMB_PARALLELS,
_strlenA(SMB_PARALLELS));

#ifdef _DEBUG
			if (IsVM == 1) DebugLog(TEXT("Parallels::SMBIOS_Parallels"));
#endif

			RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
		}
	}
	/* query Parallels on PCI bus devices */
	if ( IsVM == 0 ) {
		if ( vIsInList(VID_PRLS) != NULL ) IsVM = 1;
#ifdef _DEBUG
		if (IsVM == 1) DebugLog(TEXT("VPC::VID_PRLS"));
#endif

	}
	return IsVM;
}
```

3.5. Sandboxie detection

```
BOOL IsSandboxiePresent(
	VOID
	)
{
	BYTE IsSB = 0;
	OBJECT_ATTRIBUTES attr;
	UNICODE_STRING ustrName;
	NTSTATUS Status;
	HANDLE hObject = NULL;

	WCHAR szSandboxieMutex[MAX_PATH] = {0};

	/* first check sandboxie device */
	IsSB = (IsObjectExists(DEVICELINK, DEVICE_SANDBOXIE));

#ifdef _DEBUG
	if (IsSB == 1) DebugLog(TEXT("Sandboxie::ApiDevice"));
#endif

	if ( IsSB == 0 ) {
		/* not found or error, check sandbox object directory presence */
		RtlInitUnicodeString(&ustrName, DIRECTORY_SANDBOXIE);
		InitializeObjectAttributes(&attr, &ustrName, OBJ_CASE_INSENSITIVE, NULL, NULL);
		Status = NtOpenDirectoryObject(&hObject, DIRECTORY_QUERY, &attr);
		if (NT_SUCCESS(Status)) {
			IsSB = 1;

#ifdef _DEBUG
			DebugLog(TEXT("Sandboxie::ObjectDirectory"));
#endif

			NtClose(hObject);
		}
	}


	/* query sandboxie mutex */
	if ( IsSB == 0 ) {
		IsSB = IsMutexExist(MUTEX_SANDBOXIE);

#ifdef _DEBUG
		if (IsSB == 1) DebugLog(TEXT("Sandboxie::Mutex"));
#endif

	}
	/* query sandboxie rpc port presence*/
	if ( IsSB == 0 ) {
		IsSB = (IsObjectExists(RPCCONTROLLINK, PORT_SANDBOXIE));

#ifdef _DEBUG
		if (IsSB == 1) DebugLog(TEXT("Sandboxie::Port"));
#endif

	}
	/* query driver object, reg. rights elevation */
	if ( IsSB == 0 ) {
		IsSB = (IsObjectExists(DRIVERLINK, DRIVER_SANDBOXIE));

#ifdef _DEBUG
		if (IsSB == 1) DebugLog(TEXT("Sandboxie::Driver"));
#endif

	}
	return IsSB;
}
```

3.6. Sandboxie virtualization detection

```
BOOL IsSandboxieVirtualRegistryPresent(
	VOID
	)
{
	BYTE IsSB = 0;
	ULONG hashA, hashB;
	HANDLE hKey;
	NTSTATUS Status;
	UNICODE_STRING ustrRegPath;
	OBJECT_ATTRIBUTES obja;

	WCHAR szObjectName[MAX_PATH * 2] = {0};

	hashA = HashFromStrW(REGSTR_KEY_USER);

	RtlInitUnicodeString(&ustrRegPath, REGSTR_KEY_USER);
	InitializeObjectAttributes(&obja, &ustrRegPath, OBJ_CASE_INSENSITIVE, NULL, NULL);
	Status = NtOpenKey(&hKey, MAXIMUM_ALLOWED, &obja);
	if (NT_SUCCESS(Status)) {
		if ( QueryObjectName((HKEY)hKey, &szObjectName, MAX_PATH * 2, TRUE) ) {
			hashB = HashFromStrW(szObjectName);
			if (hashB != hashA) IsSB = 1;
		}
		NtClose(hKey);
	}
	return IsSB;
}

BOOL AmISandboxed(
	VOID
	)
{
	BYTE IsSB = 0;
	ULONG uLength = 0L;
	NTSTATUS Status;
	HANDLE hDummy;
	ULONG_PTR k, i, FileID = 0xFFFFFFFF, OurID = GetCurrentProcessId();
	PSYSTEM_HANDLE_INFORMATION HandleTable = NULL;
	SYSTEM_OBJECTTYPE_INFORMATION TypeInfo = {0};
	MEMORY_BASIC_INFORMATION RegionInfo = {0};
	WCHAR szObjectName[MAX_PATH * 2] = {0};

	__try {

		/* find sandboxie api device inside our handle table */
		hDummy = OpenDevice(L"\\Device\\Null", GENERIC_READ, &Status);
		if ( hDummy == NULL ) __leave;

		HandleTable =
(PSYSTEM_HANDLE_INFORMATION)NativeAllocateInfoBuffer(SystemHandleInformation, &uLength);
		if ( HandleTable == NULL ) __leave;

		for (k=0; k<2; k++) {
			for (i=0; i<HandleTable->NumberOfHandles; i++) {
				if ( HandleTable->Handles[i].UniqueProcessId == OurID )

					if (k == 0) {
						if ( HandleTable->Handles[i].HandleValue ==
(USHORT)hDummy ) {
							FileID = HandleTable-
>Handles[i].ObjectTypeIndex;
							break;
						}
					} else {
						if (HandleTable->Handles[i].ObjectTypeIndex == FileID
)
							if ( QueryObjectName((HANDLE)HandleTable-
>Handles[i].HandleValue, &szObjectName, MAX_PATH * 2, TRUE) ) {
								if ( strstrW(szObjectName,
VENDOR_SANDBOXIE) != NULL) {

#ifdef _DEBUG
```

```
        DebugLog(TEXT("Sandboxie::HandleTable"));
#endif

                                                                        IsSB = 1;
                                                                        break;
                                                                }
                                                        }
                                        }
                        }
                }

                /* brute-force memory to locate sandboxie injected code and locate sandboxie tag
*/
                if ( IsSB == 0 ) {
                        i = (ULONG_PTR)g_siSysInfo.lpMinimumApplicationAddress;
                        do {
                                Status = NtQueryVirtualMemory(NtCurrentProcess(), (PVOID)i,
MemoryBasicInformation,
                                        &RegionInfo, sizeof(MEMORY_BASIC_INFORMATION), &uLength);
                                if (NT_SUCCESS(Status)) {
                                        if (IsExecutableCode(RegionInfo.AllocationProtect,
RegionInfo.State)) {
                                                for (k=i; k<i+RegionInfo.RegionSize;
k+=sizeof(DWORD)) {
                                                        if (*(PDWORD)k == 'kuzt') {
                                                                IsSB = 1;

#ifdef _DEBUG

        DebugLog(TEXT("Sandboxie::MemoryTag"));
#endif

                                                                break;
                                                        }
                                                }
                                        }
                                        i += RegionInfo.RegionSize;
                                } else {
                                        i += PAGE_SIZE;
                                }
                        } while (i<(ULONG_PTR)g_siSysInfo.lpMaximumApplicationAddress);
                }

                /* abuse sandboxie sbiedll.dll bug/lame behaviour */
                if ( IsSB == 0 ) {
                        IsSB = IsSandboxieVirtualRegistryPresent();

#ifdef _DEBUG
                        if (IsSB == 1) DebugLog(TEXT("Sandboxie::VirtualRegistry"));
#endif

                }
        }
        __finally {
                mmfree(HandleTable);
                if ( hDummy != NULL ) NtClose(hDummy);
        }

        return IsSB;
}
```

3.7. Hypervisor detection

```
BOOL IsHypervisor(
	VOID
	)
{
	int CPUInfo[4] = {-1};

	/*
	   query hypervisor presence
	   http://msdn.microsoft.com/en-us/library/windows/hardware/ff538624(v=vs.85).aspx
	be aware this detection can be bogus
	*/

	__cpuid(CPUInfo, 1);
	if ((CPUInfo[2] >> 31) & 1) {
		return TRUE;
	}

	return FALSE;
}

BYTE GetHypervisorType(
	VOID
	)
{
	int CPUInfo[4] = {-1};
	char HvProductName[0x40];

	__cpuid(CPUInfo, 0x40000000);
	RtlSecureZeroMemory(HvProductName, sizeof(HvProductName));
	memcpy(HvProductName, CPUInfo + 1, 12);

	/* http://msdn.microsoft.com/en-us/library/windows/hardware/ff542428(v=vs.85).aspx */
	if (_strcmpiA(HvProductName, "Microsoft Hv") == 0) {
		return 1;
	}

	/*
http://kb.VMware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=100
9458 */
	if (_strcmpiA(HvProductName, "VMwareVMware") == 0) {
		return 2;
	}
	/* Parallels VMM ids */
	if (_strcmpiA(HvProductName, "prl hyperv") == 0) {
		return 3;
	}
	return 0;
}
```

3.8. VMDE support routines

```
PVOID GetFirmwareTable(
	PULONG pdwDataSize,
	DWORD dwSignature,
	DWORD dwTableID
	)
{
	NTSTATUS Status;
	ULONG Length;
	HANDLE hProcess = NULL;
	ULONG uAddress;
	PSYSTEM_FIRMWARE_TABLE_INFORMATION pSIF = NULL;
	SIZE_T memIO = 0;

	CLIENT_ID cid;
	OBJECT_ATTRIBUTES attr;
	MEMORY_REGION_INFORMATION memInfo;

	/* use documented GetSystemFirmwareTable instead, this is it raw implementation */
	if ( g_osVer.dwMajorVersion > 5 ) {
		Length = 0x1000;
		pSIF = (PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), 
HEAP_ZERO_MEMORY, Length);
		if (pSIF != NULL) {
			pSIF->Action = SystemFirmwareTable_Get;
			pSIF->ProviderSignature = dwSignature;
			pSIF->TableID = dwTableID;
			pSIF->TableBufferLength = Length;
			/* query if info class available and if how many memory we need  */
			Status = NtQuerySystemInformation(SystemFirmwareTableInformation, pSIF, 
Length, &Length);
			if (Status == STATUS_INVALID_INFO_CLASS || Status == 
STATUS_INVALID_DEVICE_REQUEST) {
				RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
				return NULL;
			}
			if (!NT_SUCCESS(Status) || Status == STATUS_BUFFER_TOO_SMALL) {
				RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
				pSIF = 
(PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY, Length);
				if (pSIF != NULL) {
					pSIF->Action = SystemFirmwareTable_Get;
					pSIF->ProviderSignature = dwSignature;
					pSIF->TableID = dwTableID;
					pSIF->TableBufferLength = Length;
					Status = 
NtQuerySystemInformation(SystemFirmwareTableInformation, pSIF, Length, &Length);
					if (!NT_SUCCESS(Status)) {
						RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
						return NULL;
					}
					if (ARGUMENT_PRESENT(pdwDataSize))
						*pdwDataSize = Length;
				}
			} else {
				if (ARGUMENT_PRESENT(pdwDataSize))
					*pdwDataSize = Length;
			}
		}
	} else {
		/* on pre Win2k3 SP1 systems the above info class unavailable, but all required 
information
		can be found inside csrss  memory space (stored here for VDM purposes) at few fixed 
addresses
		*/
		if ((dwSignature != FIRM) && (dwSignature != RSMB)) return NULL;

		/* we are interested only in two memory regions */
		switch ( dwTableID ) {
		case FIRM:
			uAddress = 0xC0000; /* FIRM analogue */
			break;
		case RSMB:
			uAddress = 0xE0000; /* RSMB analogue */
```

```
                break;
        default:
                return NULL;
                break;
        }

        Length = 0;
        cid.UniqueProcess = (HANDLE)CsrGetProcessId();
        cid.UniqueThread = 0;
        InitializeObjectAttributes(&attr, NULL, 0, 0, NULL);
        /* open csrss, reg. client debug privilege set */
        Status = NtOpenProcess(&hProcess, PROCESS_QUERY_INFORMATION | PROCESS_VM_READ,
&attr, &cid);
        if (NT_SUCCESS(Status)) {
                /* get memory data region size for buffer allocation */
                Status = NtQueryVirtualMemory(hProcess, (PVOID)uAddress,
MemoryRegionInformation, &memInfo, sizeof(MEMORY_REGION_INFORMATION), &memIO);
                if (NT_SUCCESS(Status)) {
                        pSIF =
(PSYSTEM_FIRMWARE_TABLE_INFORMATION)RtlAllocateHeap(RtlProcessHeap(), HEAP_ZERO_MEMORY,
memInfo.RegionSize);
                        if (pSIF != NULL) {
                                /* read data to our allocated buffer */
                                Status = NtReadVirtualMemory(hProcess, (PVOID)uAddress,
pSIF, memInfo.RegionSize, &memIO);
                                if (NT_SUCCESS(Status)) {
                                        if (ARGUMENT_PRESENT(pdwDataSize))
                                                *pdwDataSize = memInfo.RegionSize;
                                } else {
                                        RtlFreeHeap(RtlProcessHeap(), 0, pSIF);
                                        return NULL;
                                }
                        }
                }
                NtClose(hProcess);
        }
    }
    return pSIF;
}

BOOL QueryObjectName(
    HKEY hKey,
    PVOID Buffer,
    ULONG BufferSize,
    BOOL IsUnicodeCall
    )
{
    POBJECT_NAME_INFORMATION pObjName;
    NTSTATUS Status;
    ULONG ReturnLength;
    BOOL bResult;
    ULONG len;

    LPSTR StrA = NULL;
    LPWSTR StrW = NULL;

    pObjName = NULL;
    ReturnLength = 0;
    bResult = FALSE;

    if ( IsUnicodeCall ) {
        StrW = (LPWSTR)Buffer;
    } else {
        StrA = (LPSTR)Buffer;
    }

    __try {

        NtQueryObject(hKey, ObjectNameInformation, NULL, ReturnLength, &ReturnLength);

        pObjName = (POBJECT_NAME_INFORMATION)mmalloc(ReturnLength);
        if ( pObjName == NULL )
            __leave;

        Status = NtQueryObject(hKey, ObjectNameInformation, pObjName, ReturnLength, NULL);
        if (NT_SUCCESS(Status)) {
```

```
                        if ( (pObjName->Name.Buffer != NULL) && (pObjName->Name.Length > 0) ) {

                                bResult = TRUE;

                                len = (ULONG)_strlenW(pObjName->Name.Buffer);
                                if (len > BufferSize) len = BufferSize;
                                if ( IsUnicodeCall ) {
                                        _strncpyW(StrW, BufferSize, pObjName->Name.Buffer,
BufferSize);
                                } else {
                                        WideCharToMultiByte(CP_ACP, 0, pObjName->Name.Buffer, len,
StrA, len, 0, 0);
                                }
                        }
                }
        } __finally {
                if (pObjName != NULL) mmfree(pObjName);
        }
        return bResult;
}


VOID EnumPCIDevsReg(
        VOID
        )
{
        HANDLE hKey = NULL;
        DWORD dwKeySubIndex = 0;
        ULONG ResultLength = 0;
        NTSTATUS Status = STATUS_UNSUCCESSFUL;

        UNICODE_STRING ustrKeyName;
        OBJECT_ATTRIBUTES obja;
        PKEY_BASIC_INFORMATION pKeyInfo = NULL;

        PVENDOR_ENTRY entry;
        WCHAR szTempBuf[MAX_PATH] = {0};

        RtlInitUnicodeString(&ustrKeyName, REGSTR_KEY_PCIENUM);
        InitializeObjectAttributes(&obja, &ustrKeyName, OBJ_CASE_INSENSITIVE, NULL, NULL);

        __try {
                Status = NtOpenKey(&hKey, KEY_ENUMERATE_SUB_KEYS, &obja);
                if ( hKey == NULL && !NT_SUCCESS(Status)) __leave;
                do {

                        NtEnumerateKey(hKey, dwKeySubIndex, KeyBasicInformation, NULL, 0,
&ResultLength);
                        pKeyInfo = (PKEY_BASIC_INFORMATION)RtlAllocateHeap(RtlProcessHeap(),
HEAP_ZERO_MEMORY, ResultLength);
                        if ( pKeyInfo == NULL) __leave;

                        Status = NtEnumerateKey(hKey, dwKeySubIndex, KeyBasicInformation, pKeyInfo,
ResultLength, &ResultLength);
                        if (NT_SUCCESS(Status)) {
                                entry = (PVENDOR_ENTRY)mmalloc(sizeof(VENDOR_ENTRY));
                                if ( entry ) {
                                        _strncpyW(entry->VendorFullName, MAX_PATH, pKeyInfo->Name,
pKeyInfo->NameLength / sizeof(WCHAR));
                                        vExtractID(entry);
                                        InsertHeadList(&VendorsListHead, &entry->ListEntry);
                                }
                                RtlFreeHeap(RtlProcessHeap(), 0, pKeyInfo);
                                pKeyInfo = NULL;
                        }
                        dwKeySubIndex++;

                } while ( NT_SUCCESS(Status) );

        } __finally {
                if (hKey != NULL) NtClose(hKey);
                if ( pKeyInfo != NULL) RtlFreeHeap(RtlProcessHeap(), 0, pKeyInfo);
        }
}
```

N. Rin

EP_X0FF

This document converted from slides.