J Comput Virol Hack Tech (2015) 11:51–58
DOI 10.1007/s11416-014-0224-9

CORRESPONDENCE

How to detect the Cuckoo Sandbox and to Strengthen it?

Olivier Ferrand

Received: 19 February 2014 / Accepted: 26 August 2014 / Published online: 4 September 2014

Abstract Nowadays a lot of malware are analyzed with virtual machines. The Cuckoo sandbox (Cuckoo DevTeam: Cuckoo sandbox. http://www.cuckoosandbox.org, 2013) offers the possibility to log every actions performed by the malware on the virtual machine. To protect themselves and to evande detection, malware need to detect whether they are in an emulated environment or in a real one. With a few modifications and tricks on Cuckoo and the virtual machine we can try to prevent malware to detect that they are under analyze, or at least make it harder. It is not necessary to apply all the modifications, because it may produce a significant overhead and if malware checks his execution time, it may detect an anomaly and consider that it is running in a virtual machine. The present paper will show how a malware can detect the Cuckoo sandbox and how we can counter that.

Keywords Computer security · Attacks · Malware · Cuckoo Sandbox · Virtual machine · VirtualBox

1 Introduction

The Cuckoo Sandbox [1] is an open source malware analysis system. The development has started in summer 2010 in a Google Summer Code project. The actual version is 1.0. Its goal is to provide a way to analyze files automatically and to provide all the interactions between the files under analysis and the system. The main targets are Windows executables, DLL, Internet URLs, PDF, Office documents and Java files. In this paper, we present how a malware can detect whether Cuckoo is trying to analyze it or not. For each attack presented, we give the mechanism used for the detection of Cuckoo and an example of code which allows to realize that. The presented techniques are working on the 0.4 and 0.5 versions of Cuckoo. Because it is a very active project, future modifications may integrate further protections to prevent our attacks which are presented in this paper.

We will speak about solutions to avoid the detection. In some cases, the same countermeasure can be used to counter two or three attacks, so we give the most specific solution in priority.

Finally, we will discuss about the detection of the virtual machine which is hosted by Cuckoo. Actually a good proportion of malware try to detect whether they are in a virtual machine or a physical machine. With the knowledge of internal mechanisms in Cuckoo, it is possible to add some code in order to fool the malware by giving erroneous information.

O. Ferrand (✉)
Operational Cryptology and Virology Laboratory (CVO), ESIEA, Laval, France
e-mail: olivier.ferrand@esiea-ouest.fr; ferrand@esiea-ouest.fr

2 Detection of Cuckoo

In this part, we present the different techniques that may be used by malware to detect whether they are under Cuckoo analysis or not.

2.1 Hooks' detection

The analysis of the dynamic link library cuckoomon.dll source code and particulary the files *cuckoomon.c* and *hooking.c* provides information about the technical implementation of hooks. Currently the only technique used is **HOOK_JMP_DIRECT**, as shown in the following code:

```
// get a random hooking technique, except
    for "direct jmp"
// #define HOOKTYPE (1 + (random()
#define HOOKTYPE HOOK_JMP_DIRECT

void set_hooks_dll(const wchar_t *library,
     int len)
{
    for (int i = 0; i < ARRAYSIZE(g_hooks)
        ; i++) {
        if(!wcsnicmp(g_hooks[i].library,
            library, len)) {
            hook_api(&g_hooks[i], HOOKTYPE
                );
        }
    }
}

void set_hooks()
{
    // the hooks contain the gates as well
        , so they have to be RWX
    DWORD old_protect;
    VirtualProtect(g_hooks, sizeof(g_hooks
        ), PAGE_EXECUTE_READWRITE,
        &old_protect);

    hook_disable();

    // now, hook each api :)
    for (int i = 0; i < ARRAYSIZE(g_hooks)
        ; i++) {
        hook_api(&g_hooks[i], HOOKTYPE);
    }

    hook_enable();
}
```

Listing 1 Hook selection in cuckoomon.c

The function, for this type of hook, which is defined in the file *hooking.c*, is sufficiently explicit about the implementation method as shown in the following extracted code:

```
// direct 0xe9 jmp
static int hook_api_jmp_direct(hook_t *h,
    unsigned char *from,
    unsigned char *to)
{
    // unconditional jump opcode
    *from = 0xe9;

    // store the relative address from
        this opcode to our hook function
    *(unsigned long *)(from + 1) = (
        unsigned char *) to - from - 5;
    return 0;
}
```

Listing 2 Implementation in hooking.c

With this information and the list of the hooked functions, given by the table **hook_t g_hooks[]** in the file *cuckoomon.c*, it is easy to create a code whose purpose is to obtain the address of one function and check its first opcode.

```
FARPROC addr;
addr = GetProcAddress( LoadLibraryA("
    kernel32.dll"),"DeleteFileW");
if ( *(BYTE*) addr == 0xE9 ) printf("/!\\
    Hooked by cuckoo\n");
```

Listing 3 Hook detection on with the function *DeleteFileW*

2.2 Folder's detection

By default, Cuckoo uses a specific folder on the guest system in order to store and retrieve some information to the host. Under a Windows virtual machine, the default directory is *c:\cuckoo*.

```
    def _get_root(self, root="", container
        ="cuckoo", create=True):
        """Get Cuckoo path.
        @param root: force root folder,
            don't detect it.
        @param container: folder which
            will contain Cuckoo, not used
            root parameter is used.
        @param create: create folder.
        """
        global ERROR_MESSAGE

        if not root:
            if self.system == "windows":
                root = os.path.join(os.
                    environ["SYSTEMDRIVE"]
                     + os.sep, container)
            elif self.system == "linux" or
                 self.system == "darwin":
                root = os.path.join(os.
                    environ["HOME"],
                    container)
            else:
                ERROR_MESSAGE = "Unable to
                     identify operating
                    system"
                return False

        if create and not os.path.exists(
            root):
            try:
                os.makedirs(root)
            except OSError as e:
                ERROR_MESSAGE = e
                return False
        else:
            if not os.path.exists(root):
                ERROR_MESSAGE = "Directory
                     not found:
                return False

        return root
```

Listing 4 Setting up the shared folder in *agent.py*

Indeed it is relatively easy to detect it by looking for the presence of the folder with a possible code like the following one:

```
DWORD dwattrib ;
dwattrib = GetFileAttributes(L"c:\\cuckoo"
    );
if ( ( dwattrib != INVALID_FILE_ATTRIBUTES
     ) && ( dwattrib &
    FILE_ATTRIBUTE_DIRECTORY ) )
    printf("/!\\ Folder c:\\cuckoo found
        !\n");
```

Listing 5 Detecting the cuckoo's shared folder

2.3 Pipe's detection

In a similar manner we used for Cuckoo's directory, it is possible to detect the presence of the pipe used to communicate between the host system and the guest system. Since the name is hardcoded, it is very easy to create a small piece of code in order to detect the Cuckoo's communication pipe.

```
HANDLE hFind;
hFind = CreateFile(L"\\\\.\\pipe\\cuckoo",
                    GENERIC_READ |
                        GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL);
if ( hFind != INVALID_HANDLE_VALUE ){
    CloseHandle(hFind);
    printf("/!\\ Pipe \\\\.\\pipe\\cuckoo
        found !\n");
}
```

Listing 6 Detection of the pipe

2.4 Cuckoo's agent detection

Even if Python is a common programming language, it is relatively rare to find it running on a Windows client subsystem. From that, in order to prevent a detection, a few malware may try to detect the running process python.exe or pythonw.exe. In case of doubt, the malware might try to detect the presence of an installation of python package on the machine or checking a few names of directories or registry keys.

2.5 Antihooking

This trick is not a real detection one, but it is more close to a way to avoid the analysis by Cuckoo. By default Cuckoo sets 3 hooks for the creation of a new process and use them to analyse the new process. One technique consists in using two executables, the first will initially restore its API calls, then run the second. As the 3 API are restored, Cuckoo will not detect that the second executable has been launched, so this one will not be analyzed. The following code shows how to restore the original API for a Windows XP with SP2/SP3 operating system.

```
DWORD old_protect;
BYTE *op2;
BYTE *op3;
BYTE *op1;

op1 = (BYTE *) GetProcAddress(LoadLibraryA
    ("ntdll.dll"),"ZwCreateProcess");
VirtualProtect(op1, 10,
    PAGE_EXECUTE_READWRITE, &old_protect);
*(op1) = 0xb8;
*(op1+1) = 0x2f;
*(op1+2) = 0x00;
*(op1+3) = 0x00;
*(op1+4) = 0x00;

op2 = (BYTE *) GetProcAddress(LoadLibraryA
    ("ntdll.dll"),"ZwCreateProcessEx");
VirtualProtect(op2, 10,
    PAGE_EXECUTE_READWRITE, &old_protect);
*(op2) = 0xb8;
*(op2+1) = 0x30;
*(op2+2) = 0x00;
*(op2+3) = 0x00;
*(op2+4) = 0x00;

op3 = (BYTE *) GetProcAddress(LoadLibraryA
    ("kernel32.dll"),"
    CreateProcessInternalW");
VirtualProtect(op3, 10,
    PAGE_EXECUTE_READWRITE, &old_protect);
*(op3) = 0x68;
*(op3+1) = 0x08;
*(op3+2) = 0x0A;
*(op3+3) = 0x00;
*(op3+4) = 0x00;
```

Listing 7 Call restoration on Windows XP SP2/3

3 Detection of VirtualBox

Because Cuckoo is in a Virtual Machine, it is important to secure the Virtual Machine to avoid the most current detection techniques. In this part, we will discuss about two sections: the first is about the detection of VirtualBox without the Guest Additions while the second is with the Guest Additions. The guest additions are used to provide closer integration between host and guest and to improve the interactive performance of guest systems. Even if they are useful for a classical use of a virtual machine, in case of malware analysis, they generally offer an advantage to the malware.

3.1 Case without the Guest Additions installed

The first possibility is to read a few registry keys. The following codes are the main used by the malware to detect a VirtualBox guest. They read the APCI, IDE and SYSTEM keys and their subkeys in order to find relationships with VirtualBox, generally with the name *vbox* or *virtualbox*.

```
HK = 0;
char *subkey = "SYSTEM\\CurrentControlSet
    \\Enum\\IDE";
if ((ERROR_SUCCESS ==
     RegOpenKeyEx (HKEY_LOCAL_MACHINE,
         subkey, 0, KEY_READ, &HK)) && HK)
  {
    unsigned long n_subkeys = 0;
    unsigned long max_subkey_length = 0;
    if (ERROR_SUCCESS ==
    RegQueryInfoKey (HK, 0, 0, 0, &
        n_subkeys, &max_subkey_length, 0,
        0, 0,
             0, 0, 0))
      {
    if (n_subkeys)
      {
        char *pNewKey =
          (char *) LocalAlloc (
              LMEM_ZEROINIT,
              max_subkey_length + 1);
        for (unsigned long i = 0; i <
            n_subkeys; i++)
          {
        memset (pNewKey, 0,
            max_subkey_length + 1);
        HKEY HKK = 0;
        if (ERROR_SUCCESS ==
            RegEnumKey (HK, i, pNewKey,
                max_subkey_length + 1))
          {
            if ((RegOpenKeyEx (HK, pNewKey
                , 0, KEY_READ, &HKK) ==
             ERROR_SUCCESS) && HKK)
              {
            unsigned long nn = 0;
            unsigned long maxlen = 0;
            RegQueryInfoKey (HKK, 0, 0, 0,
                 &nn, &maxlen, 0, 0, 0,
                     0, 0, 0);
            char *pNewNewKey =
              (char *) LocalAlloc (
                  LMEM_ZEROINIT, maxlen +
                  1);
            if (RegEnumKey (HKK, 0,
                pNewNewKey, maxlen + 1) ==
                ERROR_SUCCESS)
              {
                HKEY HKKK = 0;
                if (RegOpenKeyEx
                (HKK, pNewNewKey, 0,
                    KEY_READ,
                 &HKKK) == ERROR_SUCCESS)
                  {
                unsigned long size = 0xFFF
                    ;
                unsigned char ValName[0
                    x1000] = { 0 };
                if (RegQueryValueEx
                    (HKKK, "FriendlyName",
                         0, 0, ValName,
                     &size) ==
                         ERROR_SUCCESS)
                  {
                    ToLower (ValName);
                    if (strstr ((char *)
                        ValName, "vbox"))
                      {
                    printf("Virtualbox
                        detected\n");
                      }
                  }
                RegCloseKey (HKKK);
                  }
              }
            LocalFree (pNewNewKey);
            RegCloseKey (HKK);
              }
          }
          }
        LocalFree (pNewKey);
      }
      }
    RegCloseKey (HK);
  }
```

Listing 8 First method

```
HK = 0;
if (RegOpenKeyEx
    (HKEY_LOCAL_MACHINE, "HARDWARE\\
        DESCRIPTION\\System", 0, KEY_READ,
     &HK) == ERROR_SUCCESS)
  {
    unsigned long type = 0;
    unsigned long size = 0x100;
    char *systembiosversion = (char *)
        LocalAlloc (LMEM_ZEROINIT, size +
        10);
    if (ERROR_SUCCESS ==
    RegQueryValueEx (HK, "
        SystemBiosVersion", 0, &type,
             (unsigned char *)
                 systembiosversion, &size)
                 )
      {
    ToLower ((unsigned char *)
        systembiosversion);
    if (type == REG_SZ || type ==
        REG_MULTI_SZ)
      {
        if (strstr (systembiosversion, "
            vbox"))
          {
        printf("VirtualBox detected\n");
          }
      }
      }
    LocalFree (systembiosversion);

    type = 0;
    size = 0x200;
    char *videobiosversion = (char *)
        LocalAlloc (LMEM_ZEROINIT, size +
        10);
    if (ERROR_SUCCESS ==
    RegQueryValueEx (HK, "VideoBiosVersion
        ", 0, &type,
             (unsigned char *)
                 videobiosversion, &size))
      {
    if (type == REG_MULTI_SZ)
      {
        char *video = videobiosversion;
        while (*(unsigned char *) video)
          {
        ToLower ((unsigned char *) video);
        if (strstr (video, "oracle") ||
            strstr (video, "virtualbox"))
          {
            printf("VirtualBox detected\n"
                );
          }
        video = &video[strlen (video) +
            1];
          }
      }
      }
    LocalFree (videobiosversion);
    RegCloseKey (HK);
  }
```

Listing 9 Second method

The second technique consists in looking for the shared folders having a specific name such as *VirtualBox Shared Folders*. It can be done with the following code:

```
unsigned long pnsize = 0x1000;
char *provider = (char *) LocalAlloc (
    LMEM_ZEROINIT, pnsize);
int retv = WNetGetProviderName (
    WNNC_NET_RDR2SAMPLE, provider, &pnsize
    );
if (retv == NO_ERROR)
  {
    if (lstrcmpi (provider, "VirtualBox
        Shared Folders") == 0)
      {
    printf("VirtualBox detected\n");
      }
  }
```

Listing 10 Third method

3.2 Case with the Guest Additions installed

In this subsection, we discuss the detection techniques of VirtualBox using the guest additions. The first one deals with finding the *VBoxMiniRdrDN* driver. This one is used to create shared folders between the host and the guest machine.

```
HANDLE hF1 = CreateFile ("\\\\.\\
    VBoxMiniRdrDN", GENERIC_READ,
             FILE_SHARE_READ |
                 FILE_SHARE_WRITE |
             FILE_SHARE_DELETE, 0,
             OPEN_EXISTING, 0, 0);
if (hF1 != INVALID_HANDLE_VALUE)
  {
    printf ("VirtualBox detected");
  }
```

Listing 11 Shared folder detection

We can detect although the presence of a few registry keys and the Dynamic Link Library *VBoxHook.dll* which is used to load the different drivers for the Guest Addition.

In the spirit of the previous method we used on Cuckoo, we can detect the presence of a pipe with the specific name of the system tray which has been created by the add-on for the guest.

```
HANDLE hxx = CreateFile ("\\\\.\\pipe\\
    VBoxTrayIPC", GENERIC_READ,
        FILE_SHARE_READ | FILE_SHARE_WRITE
        , 0, OPEN_EXISTING, 0, 0);
if (hxx != INVALID_HANDLE_VALUE)
  {
    printf ("VirtualBox detected\n");
  }
```

Listing 12 Pipe detection

Another way consists in detecting any GUI process with a specific name with respect to the VirtualBox tray.

```
HWND hY1 = FindWindow ("
    VBoxTrayToolWndClass", 0);
HWND hY2 = FindWindow (0, "VBoxTrayToolWnd
    ");
if (hY1 || hY2)
  {
    printf ("VirtualBox detected\n");
  }
```

Listing 13 GUI detection

4 Countermeasures

In this section, we will speak about the different countermeasures which can be used in order to perform more stealth analysis, without being detected by malware. The first subsection relates to Cuckoo while the second deals with VirtualBox. A few of these countermeasures can be used with Cuckoo and VirtualBox.

4.1 Cuckoo

According to AlienVault [3], it is possible to modify the *cuckoomon.dll* file. But this way may have performance impacts for a deep analysis because we have to check each request towards the registry table. Indeed if we try to analyze a process which manipulates a huge amount of registry keys, Cuckoo checks all the keys and compare those keys with different values. The same issue remains unchanged for files, even if they are less used.

4.1.1 Using cuckoomon.dll

To avoid some accesses to special keys, it is possible to edit the file *hook_file.c* and to add directly some specific response that we want to see returned to the malware. For example, we can modify the following code:

```
HOOKDEF(LONG, WINAPI, RegOpenKeyExA,
  __in        HKEY hKey,
  __in_opt    LPCTSTR lpSubKey,
  __reserved  DWORD ulOptions,
  __in        REGSAM samDesired,
  __out       PHKEY phkResult
) {
    LONG ret = Old_RegOpenKeyExA(hKey,
        lpSubKey, ulOptions, samDesired,
        phkResult);
    LOQ("psP", "Registry", hKey, "SubKey",
         lpSubKey, "Handle", phkResult);
    return ret;
}
```

Listing 14 Original code

into:

```
HOOKDEF(LONG, WINAPI, RegOpenKeyExA,
  __in        HKEY hKey,
  __in_opt    LPCTSTR lpSubKey,
  __reserved  DWORD ulOptions,
  __in        REGSAM samDesired,
  __out       PHKEY phkResult
) {
    LONG ret;
    if ((strstr(lpSubKey, "VirtualBox") !=
         NULL) || (strstr(lpSubKey, "VBox"
        ) != NULL) ) {
        ret = 1;
        LOQ("s", "Blocked Registry key", "
            RegOpenKeyExA");
    }
    else {
        ret = Old_RegOpenKeyExA(hKey,
            lpSubKey, ulOptions,
            samDesired,
            phkResult);
    }
    LOQ("psP", "Registry", hKey, "SubKey",
         lpSubKey, "Handle", phkResult);
    return ret;
}
```

Listing 15 Modified code

Whenever the malware try to access to the subkey VirtualBox or VBox with the RegOpenKeyExA API, Cuckoo will log that and will return that the key does not exist. In the same way, we have to modify the RegQueryValueExA hook, in order to block access to some keys.
By default, Cuckoo does not log the GetFileAttributesA API, so we must add it to the source file. Using the msdn documentation [2] we can write the following code in order to log and bypass detection mechanisms from the malware:

```
HOOKDEF(DWORD, WINAPI, GetFileAttributesA,
  __in      LPCTSTR lpFileName
) {
    if (strstr(lpFileName, "cuckoo") !=
        NULL) {
        LOQ("s", "Blocked File access", "
            GetFileAttributesA");
        return INVALID_FILE_ATTRIBUTES;
    }
    else
        return Old_GetFileAttributesA(
            lpFileName);
}
```

Listing 16 New hook's code

Technically it is easy to add new hooks. For any new technique we can create a new countermeasure by adding a hook in the dll. But it takes time to react to each new attack, mainly when it depends from the number of operations performed by malicious codes. In addition, if the malware compute execution time, it may guess that it is under analysis. The quantity of logs can be too huge to allow a simple analysis.

4.1.2 Modifying Cuckoo directly

Another approach consists in modifying directly Cuckoo. Since Cuckoo is an opensource project, we can modify the *agent.py* file in order to change the name of the directory used for the analysis. We can use a more common name such as "windows_" or "nt" or anything else we want. So the malware will not be able to guess whether it is running with Cuckoo or not, because it will not be able to find fixed file/registry names.
For the pipe, the same system is used, but this time we have to modify the core of Cuckoo although.
The last modification that we can perform on Cuckoo is to compile the file *agent.py*, in order to produce an executable. We can use **py2exe** to perform this action. It relies on the fact that there are not real reason to find the processes *python.exe* or *pythonw.exe* running in a classical computer. So if the agent is in the form of an executable, the malware will see another process like or among many others. By doing that, the python package is no longer necessary on the guest machine.

4.2 VirtualBox

Because strengthening Cuckoo is not the only possible solution, we have to modify our virtual machines although. Some of the previous actions can be done directly with the VirtualBox Manager [4]. One of the first thing to do is to anonymize the hardware on the virtual machine. The best way to do this, is to not install the VirtualTools on the guest machine because it creates a lot of files on the system and many of them can be detected easily. The rest of the work can be divided in two parts.

4.2.1 Registry

Virtualbox creates a lot of registry keys. Those keys can be removed easily with a small batch file. Before removing those keys, it is better to copy them under another name without any reference to VirtualBox. Some keys can only be removed in the safe boot mode of Windows or require the system permission rights. The file can be as the following one:

```
@reg copy HKLM\HARDWARE\ACPI\DSDT\VBOX__
    HKLM\HARDWARE\ACPI\DSDT\backup__ /s /f
@reg delete HKLM\HARDWARE\ACPI\DSDT\VBOX__
     /f
@reg copy HKLM\HARDWARE\ACPI\RSDT\VBOX__
    HKLM\HARDWARE\ACPI\RSDT\backup__ /s /f
@reg delete HKLM\HARDWARE\ACPI\RSDT\VBOX__
     /f
@reg copy HKLM\HARDWARE\ACPI\FADT\VBOX__
    HKLM\HARDWARE\ACPI\FADT\backup__ /s /f
@reg delete HKLM\HARDWARE\ACPI\FADT\VBOX__
     /f

@reg copy HKEY_LOCAL_MACHINE\HARDWARE\ACPI
    \DSDT\backup__\VBOXBIOS
HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\
    backup__\myBIOS /s /f
@reg delete HKEY_LOCAL_MACHINE\HARDWARE\
    ACPI\DSDT\backup__\VBOXBIOS /f
@reg copy HKEY_LOCAL_MACHINE\HARDWARE\ACPI
    \FADT\backup__\VBOXFACP
HKEY_LOCAL_MACHINE\HARDWARE\ACPI\FADT\
    backup__\myFACP /s /f
@reg delete HKEY_LOCAL_MACHINE\HARDWARE\
    ACPI\FADT\backup__\VBOXFACP /f
@reg copy HKEY_LOCAL_MACHINE\HARDWARE\ACPI
    \RSDT\backup__\VBOXRSDT
HKEY_LOCAL_MACHINE\HARDWARE\ACPI\RSDT\
    backup__\myRSDT /s /f
@reg delete HKEY_LOCAL_MACHINE\HARDWARE\
    ACPI\RSDT\backup__\VBOXRSDT /f

@reg add HKEY_LOCAL_MACHINE\HARDWARE\
    DESCRIPTION\System /v
SystemBiosVersion /t REG_MULTI_SZ /d "
    backup -1" /f
@reg add HKEY_LOCAL_MACHINE\HARDWARE\
    DESCRIPTION\System /v
VideoBiosVersion /t REG_MULTI_SZ /d "
    VGABIOS 1.0" /f
```

Listing 17 Sample of batch file

The key names may depend on the operating system or the version number of VirtualBox.

4.2.2 Hardware

Since the version 4.2 of VirtualBox, it is possible to modify some hardware part with the VirtualBox Manager. The **hdparm** command give us enough information to configure correctly the hard drive and the CDROM drive.

```
root@VxLab:~# hdparm -i /dev/sda

/dev/sda:

 Model=ST2000DM001-9YN164, FwRev=CC4B,
     SerialNo=W1E1CVJX
 Config={ HardSect NotMFM HdSw>15uSec
     Fixed DTR>10Mbs RotSpdTol>.5 }
```

```
  RawCHS=16383/16/63, TrkSize=0, SectSize
      =0, ECCbytes=4
  BuffType=unknown, BuffSize=unknown,
      MaxMultSect=16, MultSect=16
  CurCHS=16383/16/63, CurSects=16514064,
      LBA=yes, LBAsects=3907029168
  IORDY=on/off, tPIO={min:120,w/IORDY:120},
        tDMA={min:120,rec:120}
  PIO modes:  pio0 pio1 pio2 pio3 pio4
  DMA modes:  mdma0 mdma1 mdma2
  UDMA modes: udma0 udma1 udma2 udma3 udma4
        udma5 *udma6
  AdvancedPM=yes: unknown setting
      WriteCache=enabled
  Drive conforms to: unknown:  ATA/ATAPI
      -4,5,6,7

  * signifies the current active mode

root@VxLab:~#
```

Listing 18 Getting the hard drive information

We can change some information of the hard drive and the CDROM drive using the information from the previous command.

```
VBoxManage setextradata "<vmname>" "
    VBoxInternal/Devices/piix3ide/0/Config
    /PrimaryMaster/SerialNumber"  "
    SerialNo"
VBoxManage setextradata "<vmname>" "
    VBoxInternal/Devices/piix3ide/0/Config
    /PrimaryMaster/FirmwareRevision" "<
    FwRev>"
VBoxManage setextradata "<vmname>" "
    VBoxInternal/Devices/piix3ide/0/Config
    /PrimaryMaster/ModelNumber" "<Model>"
```

Listing 19 Changing the hard drive information for the PIIX3 controlor

By default the MAC address is directly tagged as an Oracle card. In order to prevent a detection by the analysis of the network card by a malware, we can change its prefix by an other constructor.

```
 VBoxManage modifyvm "<vmname>" --
     macaddressX <MAC>
```

The real information of the computer can be obtained with the command : **dmidecode**. The following extract shows which information can be changed:

```
### Bios information (-t 0)
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSVendor"        "
          Vendor"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSVersion"       "
          Version"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSReleaseDate"   "
          Release Date"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSReleaseMajor"  X
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSReleaseMinor"  X
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSFirmwareMajor" X
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBIOSFirmwareMinor" X

### System Information (-t 1)
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemVendor"      "
          Manufacturer"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemProduct"     "
          Product Name"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemVersion"     "
          Version"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemSerial"      "
          Serial Number"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemSKU"         "
          SKU Number"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemFamily"      "
          Family"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiSystemUuid"        "
          UUID"

### Base Board Information (-t 2)
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardVendor"       "
          Manufacturer"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardProduct"      "
          Product Name"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardVersion"      "
          Version"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardSerial"       "
          Serial Number"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardAssetTag"     "
          Asset Tag"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardLocInChass"   "
          Location in Chassis"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiBoardType"         10
          ## MotherBoard

### Chassis Information (-t 3)
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiChassisVendor"     "
          Manufacturer"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiChassisVersion"    "
          Version"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
          Config/DmiChassisSerial"     "
          Serial Number"
```

```
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
         Config/DmiChassisAssetTag"    "
         Asset Tag"
VBoxManage setextradata "VM name"

### Processor Information (-t 4)
      "VBoxInternal/Devices/pcbios/0/
         Config/DmiProcManufacturer"  "
         Manufacturer"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
         Config/DmiProcVersion"       "
         Version"
VBoxManage setextradata "VM name"

### OEM Strings (-t 11)
      "VBoxInternal/Devices/pcbios/0/
         Config/DmiOEMVBoxVer"        "
         Version"
VBoxManage setextradata "VM name"
      "VBoxInternal/Devices/pcbios/0/
         Config/DmiOEMVBoxRev"        "
         Revision"
```

Listing 20 DMI information to change

A good idea is to change the original BIOS code with the real BIOS code of the host computer. With this modification the behavior will be really close to a real machine hosting.

```
dd if=/sys/firmware/acpi/tables/SLIC of=
    SLIC.bin
VBoxManage setextradata "<VM name>" "
    VBoxInternal/Devices/acpi/0/Config/
    CustomTable" SLIC.bin
```

Listing 21 Changing the BIOS

5 Conclusion

In this document, we have shown how to detect Cuckoo and how we can prevent its detection by strengthening Cuckoo. More generally we can configure a virtual machine with VirtualBox in a closer way. In all cases, Cuckoo remains reliable and advanced enough to perform automatic and relatively comprehensive analyses of malware. With this hardening it becomes more difficult for malware to bypass a virtual machine equipped with the Cuckoo analysis system. The main problem is to find the best trade-off between performance and the time used to carry out such an analysis.

References

1. Cuckoo DevTeam: Cuckoo sandbox (2013). http://www.cuckoosandbox.org
2. msdn: Getfileattributes function (2013). http://msdn.microsoft.com/en-us/library/windows/desktop/aa364944(v=vs.85).aspx
3. Ortega, A.: Hardening cuckoo sandbox against vm aware malware (2012). http://labs.alienvault.com/labs/index.php/2012/hardening-cuckoo-sandbox-against-vm-aware-malware/
4. VirtualBox: Virtualbox manual (2013). http://www.virtualbox.org/manual/