How to detect the Cuckoo Sandbox and hardening it?

Olivier FERRAND
ESIEA : Operational Cryptology and Virology Laboratory (CVO)
ferrand@esiea-ouest.fr

About Authors

Olivier Ferrand is a Ph.D. student in Computer Science, specialisation: Computer Security. He works at the Operational Cryptology and Virology lab. His research focuses on security of computer systems and virology defense. E-mail: olivier.ferrand@esiea-ouest.fr

Keywords

Computer Security, Attacks, Malware, Cuckoo Sandbox, Virtual Machine, VirtualBox
Abstract

Actually lot of malwares are analyzed with via virtual machines. The sandbox Cuckoo offer us the possbility to log every actions done by the malware on the virtual machine, but many malwares try to detected if they are in a emulator or in a real machine. With some modifications on cuckoo and the virtual machine, which is supposed be VirtualBox, the malwares do not detect that they are running in a protected system and are logged totaly.

It is not necessary to apply all the modifications, because it can produce a significant overhead and if malware checks his execution time, it can detect an anomaly and consider that it is running in a virtual machine. The present document will show, how we can detect the Cuckoo sandbox and how we can counter that.

Introduction

Cuckoo Sandbox[1] is a malware analysis system. The development started in 2011 and the actual version is 0.5. This system permits to analyze what the malware do with the operating system by logging every actions done.

In this paper, we present how a malware can detect if cuckoo is trying to analyze it. For each attack presented, we give the mechanism used for the detection of Cuckoo and an example of code which permits 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, futures modifications may integrate some protections against it.

In the second part of the document we speak about solutions to avoid the detection. In some cases, the same possibility can be used to counter to or three attacks, so we give the most specific solution in priority.

In the third part, we discuss about the detection of the virtual machine which is hosted by Cuckoo. Actually lot of malware try to detect if they are in a virtual machine or a physical machine, With knowledge of the mechanisms of operation of cuckoo, it is possible to add some instructions in order to fool the malware by giving erroneous information.
1 Detection of Cuckoo

In this part we speak about the different techniques that can be used by malwares to detected if they are spied by Cuckoo or not.

1.1 Hooks’ detection

The analysis of the dynamic link library cuckoomon.dll source code and particulary the files cuckoomon.c and hooking.c give us information about the technical implemention of hooks. Currently the only technique used is HOOK_JMP_DIRECT, as shows the following code:

```
// get a random hooking technique, except for "direct jmp"
// #define HOOKTYPE (1 + (random() % (HOOK_MAXTYPE - 1)))
#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, defined in the file hooking.c is sufficiently explicit about the implementation method as shown in the extract 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 which has the goal 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

1.2 Folder's detection

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

```python
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 in 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):
```
Listing 4: Setting up the shared folder in `agent.py`

So it is relatively easy to detect it by testing for the presence of the folder with a code similar to the following:

```c
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

### 1.3 Pipe’s detection

As well as for the detection of that directory, it is possible to detect the presence of the pipe used for communication between the host system and the guest system. The following code shows us some information about the pipe and especially its name.

```c
//
// Pipe API
//
// The following Format Specifiers are available:
// z -> (char *) -> zero-terminated ascii string
// Z -> (wchar_t *) -> zero-terminated unicode string
// s -> (int, char *) -> ascii string with length
// S -> (int, wchar_t *) -> unicode string with length
// o -> (UNICODE_STRING *) -> unicode string
// O -> (OBJECT_ATTRIBUTES *) -> wrapper around unicode string
// d -> (int) -> integer
// x -> (int) -> hexadecimal integer
//
int pipe(const char *fmt, ...);
int pipe2(void *out, int *outlen, const char *fmt, ...);

#define PIPE_MAX_TIMEOUT 10000
#define PIPE_NAME "\\\\\pipe\\cuckoo"
```

Listing 6: Content of `pipe.h`

Because the name is hardcoded, it is very easy to create a small piece of code in order to detect the pipe for cuckoo.
### 1.4 Agent’s detection

Even if python is a common software, it is relatively rare to found it running on a computer with Windows. So in order to prevent a detection, some malwares can try to detect the process python.exe or pythonw.exe with a similar code as following:

```c
HANDLE hProcessSnap;
PROCESSENTRY32 pe32;
hProcessSnap = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS, 0 );
if( hProcessSnap == INVALID_HANDLE_VALUE )
{
    printf("CreateToolhelp32Snapshot (of processes)"");
}
pe32.dwSize = sizeof( PROCESSENTRY32 );
if( !Process32First( hProcessSnap, &pe32 ) )
{
    printf("Process32First");
    CloseHandle( hProcessSnap );
}
do {
    if (( pe32.szExeFile[0]=='p' ) &&
        ( pe32.szExeFile[1]=='y' ) &&
        ( pe32.szExeFile[2]=='t' ) &&
        ( pe32.szExeFile[3]=='h' ) &&
        ( pe32.szExeFile[4]=='o' ) &&
        ( pe32.szExeFile[5]=='n' ) &&
        printf("/!/\ agent found !\n");
        break;
    }
} while ( Process32Next( hProcessSnap, &pe32 ) );
CloseHandle( hProcessSnap );
```

Listing 7: Detection of the pipe
1.5 Original calls restoration

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

```c
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 9: Call restoration on Windows XP SP2/3
2 Detection of VirtualBox

Because Cuckoo is running in a Virtual Machine, it is important to secure the Virtual Machine. In this part, we discuss about two sections, the first is about the detection of VirtualBox without the Guest Additions, the second is with the Guest Additions.

2.1 Without the Guest Additions installed

The first possibility is to read few registry keys. The following codes are the principally used by the malwares to detect a VirtualBox guest. They read the ACPI, IDE and SYSTEM keys and their subkeys in order to found, some relation with VirtualBox, generally with the name `vbox` and `virtualbox`.

```c
HKEY HK=0;

if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,"HARDWARE\ACPI\DSDT\VBOX_",0,KEY_READ,&HK)==ERROR_SUCCESS)
{
    printf("VirtualBox detected\n");
}
```

Listing 10: First method

```c
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;
    RegQueryInfoKey(HK, 0, 0, 0, &n_subkeys, &max_subkey_length, 0, 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,
```
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 = 0xFFFF;
        unsigned char ValName[0x1000] = { 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 11: Second 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"))
            {
The second technique is to look for shared folders with a specific name as VirtualBox Shared Folders. It can be done with the following code:

```c
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 13: Fourth method

### 2.2 With the Guest Additions installed

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

```c
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 14: First method

We can detect too the Dynamic Link Library VBoxHook.dll which is used to load the different drivers for the Guest Additions.

```c
HMODULE hM1 = LoadLibrary ("VBoxHook.dll");
if (hM1)
{
    printf ("VirtualBox detected");
}
```

Listing 15: Second method

```c
HK = 0;
if ((ERROR_SUCCESS ==
    RegOpenKeyEx (HKEY_LOCAL_MACHINE,
                  "SOFTWARE\Oracle\VirtualBox Guest Additions", 0,
                  KEY_READ, &HK)) && HK)
{
    printf ("VirtualBox detected\n");
}
```

Listing 16: Third method

As for Cuckoo, we can detect the presence of a pipe with the specific name of the system tray created by the add-on for the guest.

```c
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 17: Fourth method

An other way is to detect any window with a specific name relative to the VirtualBox tray.

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

Listing 18: Fifth method

3 Countermeasures

In this section we will speak about the different countermeasures that can be used in order to perform the best analysis. The subsection concerns Cuckoo and the second VirtualBox. Some of countermeasures can be used with Cuckoo and VirtualBox.

3.1 Cuckoo

According to AlienVault[3], it is possible to modify the cuckoomon.dll file. But this way can be very slow for a deep analysis because we have to check each request to the registry table. Indeed if we try to analyze a file which requires a huge amount of registry keys, cuckoo checks all the keys and compare thoses keys with different values. The problematic remains unchanged for files, even if they are less used.

3.1.1 Using cuckoomon.dll

To avoid some access to special keys, it is possible to edit the file hook_file.c and add directly response that we want to return 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 );
    LOG("psP", "Registry", hKey, "SubKey", lpSubKey, "Handle", phkResult);
    return ret;
}
```

Listing 19: Original code

into:
So if 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 exists. 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\[2\] we find that the syntax is the following :

```c
DWORD WINAPI GetFileAttributesA( 
_In_  LPCTSTR lpFileName
);
```

We can write the following code in order to log and fake the malware :

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

Listing 20: Hardening code

Listing 21: msdn syntax

Listing 22: New hook’s code
As it is easy to add new hooks, whenever we encounter a new technique we can create a new countermeasure with the dll. But, it takes a lot of time depending on the number of operations performed by malicious code and if the malware calculates the time, it can detect an attempt to analyze or the log can be to voluminous to permit a simple analyze.

3.1.2 Modifying Cuckoo directly

The other way consist to modify directly Cuckoo. Because Cuckoo is open-source, we can modify the agent.py file to change the name of the directory used for the analyse. We can use a more common name as windows, or nt or what we want. So the malware will not be able to say if it is running with Cuckoo or not.

For the pipe, the same system is used, but this time we have to modify the core of Cuckoo too.

The last modification that we can give to Cuckoo is to compile the file agent.py. We can use py2exe to do that. The reason for such action is that there are not real reason to find the process 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 an other process like hundred others.

3.2 VirtualBox

Because hardening Cuckoo is not the only solution, we have to hardening our virtual machines too. 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 of the machine. The best action is to not install the VirtualTools on the guest machine, because it create lot of file on the system and some of them can detected easily. The rest of the work can be split on two parts.

3.2.1 Registry

Virtualbox creates lot of registry keys, thoses keys can be remove easily with a small batch file. Before removing the keys, it is better to copy them under an other name without any reference to VirtualBox. Some keys can only be remove with the safe boot or the system permission. The file can be like the following :
The key names may vary depending on the operating system or the version of VirtualBox.

### 3.2.2 Hardware

Since the version 4.2 it is possible to modify some hardware part with the VirtualBox Manager. We can change some information of the hard drive and the CDROM drive.

```
VBoxManage setextradata "<vmname>" "VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/SerialNumber" "<serial>" 
VBoxManage setextradata "<vmname>" "VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/FirmwareRevision" "<firmware>" 
VBoxManage setextradata "<vmname>" "VBoxInternal/Devices/piix3ide/0/Config/PrimaryMaster/ModelNumber" "<model>"
```

Listing 24: Changing the hard drive information

We can change the mac address in order to obtain the same characteristics as a real card.

```
VBoxManage modifyvm "<vmname>" --macaddressX <MAC>
```
The last part to do is to change the different information about the bios.
The real information of the computer can be obtained with the command:

```bash
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSVendor" "BIOS Vendor"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSVersion" "BIOS Version"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseDate" "BIOS Release Date"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseMajor" 1
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSReleaseMinor" 2
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSFirmwareMajor" 3
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBIOSFirmwareMinor" 4
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemVendor" "System Vendor"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemProduct" "System Product"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemVersion" "System Version"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemSerial" "System Serial"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemSKU" "System SKU"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemFamily" "System Family"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiSystemUuid" "9852bf98-b38c-49db-a8de-182c42c726b"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBoardVendor" "Board Vendor"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBoardProduct" "Board Product"
VBoxManage setextradata "VM name" "VBoxInternal/Devices/pcbios/0/Config/DmiBoardVersion" "Board Version"
VBoxManage setextradata "VM name"
```
A good idea is to change the original bios with the real BIOS of the host computer. With this modification the behavior will be really near of a real machine.

Listing 25: DMI information to change

Listing 26: Changing the BIOS

Conclusion

In this document we showed how to detect Cuckoo and how we can prevent that by hardening Cuckoo and more generally how to harden a virtual machine with VirtualBox. In all cases, Cuckoo remains reliable and enough
advanced to perform automatic and relatively complete analysis of malwares.
With this hardening, it becomes more difficult for malware to bypass a vir-
tual machine. The main problem is to find the best compromise between
performance and the time used to carry such an analysis.

References
2013.


2013.