Sandbox detection

February 27, 2018

Andrei Kazakov, Sergey Grebennikov, Anas Iskhakov
Secure Systems & Network Engineering
Innopolis University

Introduction

The concept of the **Sandbox malware** analysis is to use a sample of the malware in a testing environment (**sandboxes**), where its behavior can be easily and deeply studied and analyzed without any damage to a production environment. At the same time, cybercriminals develop more sophisticated malware, finds a new way to avoid sandbox detection methods. In the last years, only a few viruses could evade detection. Today, the number of them have been increased rapidly.

1 What are common evasion techniques?

The virtual machine (**VM**) software is designed to simulate the hardware functions of real hardware, but some artifacts may be founded by malware. These artifacts can be specific files, processes, registry keys, services, network device adapters, etc. Most sandbox-evasion techniques fall into one of three categories:

- **Human interaction**.
  This category includes malicious software that requires certain actions on the part of the user - a sign that the malicious code is running on a real **PC**, and not in automatic isolated software. For example, we assume that after a malware starts a mouse pointer should move, and if it happens (check it comparing the coordinates of the pointer after a certain period of time.) malware understands that it is not in the sandbox.

- **Extended sleep**.
  Malware in this category takes advantage of the inherent constraints of file-based sandboxes. For example, knowing that a sandbox can spend for example five minutes running suspicious code for analysis, malware waits for a longer period. If the code is still running after that, it's probably not in a sandbox.

- **Environment specific**.
  Malware checks that it probably works in widely used **VM** environments. It checks for files unique to **VMware**, connected modules in the address space of the program, or **VM**-specific hardware drivers. Malware can retrieve this info in multiple ways like: **WMIC**, **Win API** and **CMD**.

  – **Vmware** - vmtoolsd.exe, vmwaretrat.exe, vmwareuser.exe, vmacthlp.exe;

  – **VirtualBox** - vboxservice.exe, vboxtray.exe.

  The next methods are also useful but they may be not accurate. The more triggering of these methods, the more likely that malware works in an isolated environment. Isolated environments often decrease the number of processors to not take up all of the computer's resources. For example, a sandbox can emulate a single-core processor. But nowadays even in mobile gadgets can be more processors. A small amount of **RAM** or small size of hard disk space also may show that a malware works in a virtual environment. Sandboxes are often very resource-demanding and often slow down the program. Malware can use timing attacks in order to understand whether hardware emulation is used or not. One option is to perform some kind of system call, that on a normal operating system works instantly and in the emulated environment will take more time because of the sandbox driver. If the difference is large, then malware assumes that it is used in a sandbox.

- **Obfuscation**
  The fourth and final most common evasion technique is obfuscating internal data. Malware that implements this tactic might use any number of tricks to run code that cannot be detected by the analysis system.

2 Can you circumvent running in the Cuckoo sandbox?

Yes, it is possible. For example, you can insert a module into the malicious code that serves to discover the **Cuckoo** agent:

```
bool processTools(string process, int size)
{
        bool flag = false;
        HANDLE hProcessSnapshot;
        PROCESSENTRY32 pe32;
        hProcessSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
        pe32.dwSize = sizeof (PROCESSENTRY32);
        if (!Process32First(hProcessSnapshot, &pe32)) CloseHandle(hProcessSnapshot);
        do
        {
                string agent = "";
                for (size_t i = 0; i < size; i++)
                {
                        agent += pe32.szExeFile[i];
                }
                if (agent == process)
                {
                        flag = true;
                        break;
                }
```

```
        } while (Process32Next(hProcessSnapshot, &pe32));
        CloseHandle(hProcessSnapshot);
        return flag;
}

void agent()
{
        if (processTools("python.", 7))
        {
                createAndWriteFile("agent.txt");
                printf("Cuckoo Detected (agent.py)\n");
        }

        if (processTools("pythonw", 7))
        {
                createAndWriteFile("agent.txt");
                printf("Cuckoo Detected (agent.py)\n");
        }

}
```

The result of the above code can be found in the Figure 1:

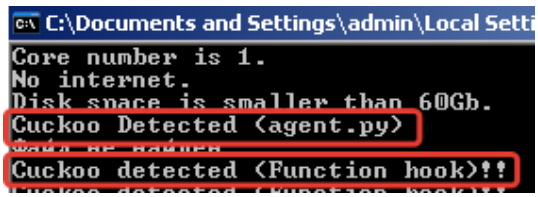

Figure 1: Detection result

But the problem is that is a very difficult task to bypass the detection of malicious code in the **Cuckoo** environment. Virtual environment Cuckoo captures an attempt to detect **Cuckoo Sandbox** through the presence of a file (Figure 5).

3 How can you react to this?

For some of the common evasion techniques we can find a way to detect a malware.

- The first category of sandbox evasion is one of the trickiest to counter. To fool sandbox-detecting malware, some vendors simulate mouse movement and clicks in their virtual-machine environments. But at the same time malware authors are upping the ante with even more sophisticated sandbox detection and evasion techniques.
- To circumvent long sleeps we can use these two techniques:

- Increase the analysis time (do long-term malware analysis). However, this is not really realistic. It simply ties up sandbox resources.

- Dynamically modify the sleep duration. This is more realistic. The sandbox will detect through analysis or debug the malware that it has multiple or long sleep commands. It

will dynamically manipulate this in the malware, manipulate it's own system clock to make the malware think it is being executed for a longer period of time, or use some sort of combination.

- In case of specific **VM**-environment we can try to hide artifacts that are related only to specific virtual machines, or on the other hand, add fake files that can be used only in the normal operating systems.

4 Results

In this lab was created simple detection software that able to detect virtual environment that is running it. This program is able to detect **VirtualBox** environment as well as the cuckoo environment. As could be seen on Figure 2 and Figure 3 program is able to detect virtual environment on **VirtualBox**. This detection used several ways such as detecting specific hardware drivers and special libraries.

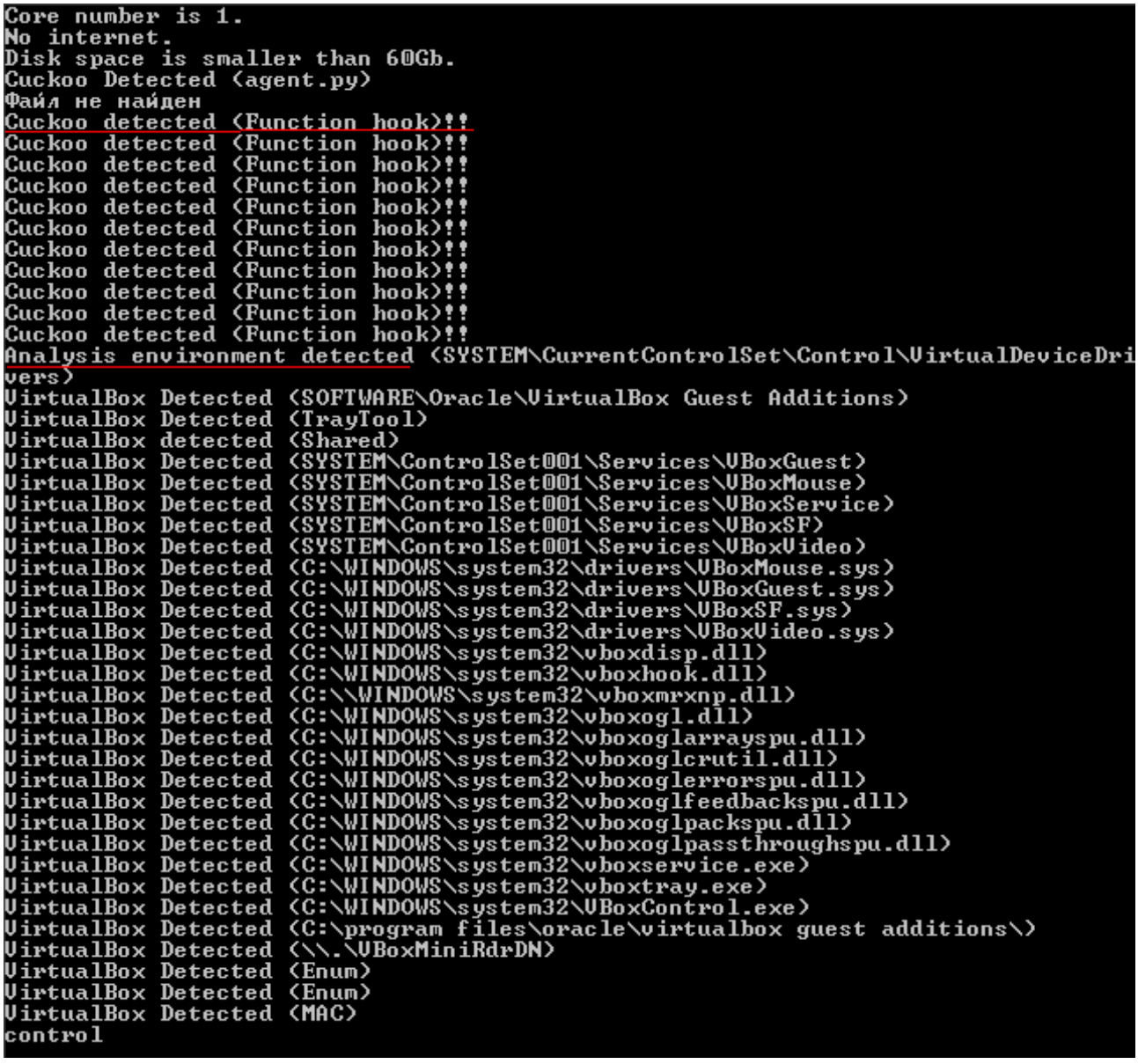

Figure 2: Detection of cuckoo virtual environment

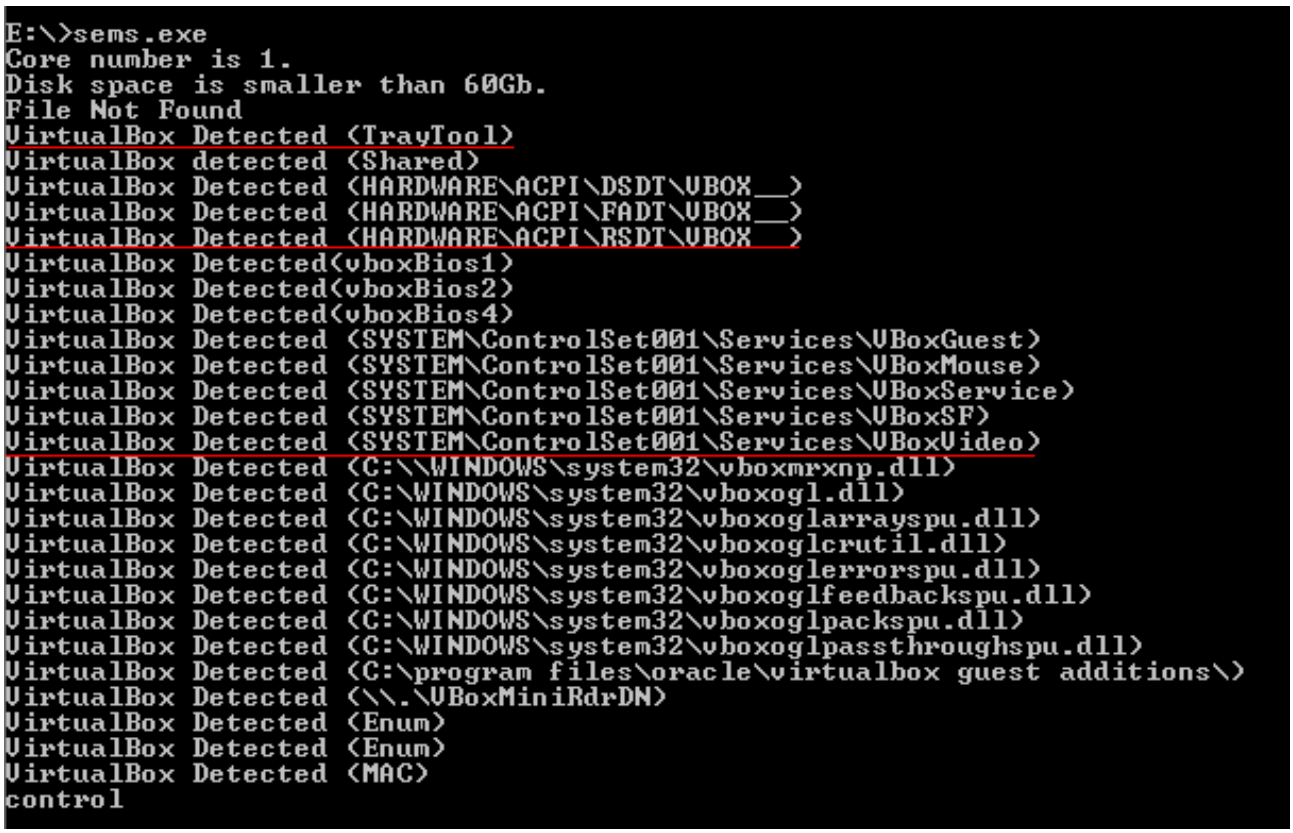

Figure 3: VirtualBox detection

The results of this program work are quite different. Program successfully detect **VirtualBox**. Bot with **Cuckoo** results is more different. First check (Figure 4) shows that our program score is **2** out of **10**. But after customization of the detection configurations (Figure 5) is shows score **14.4** out of **10**. As the conclusion could be made the result that cuckoo default detection is quite weak and should be configured manually for each kind of program.

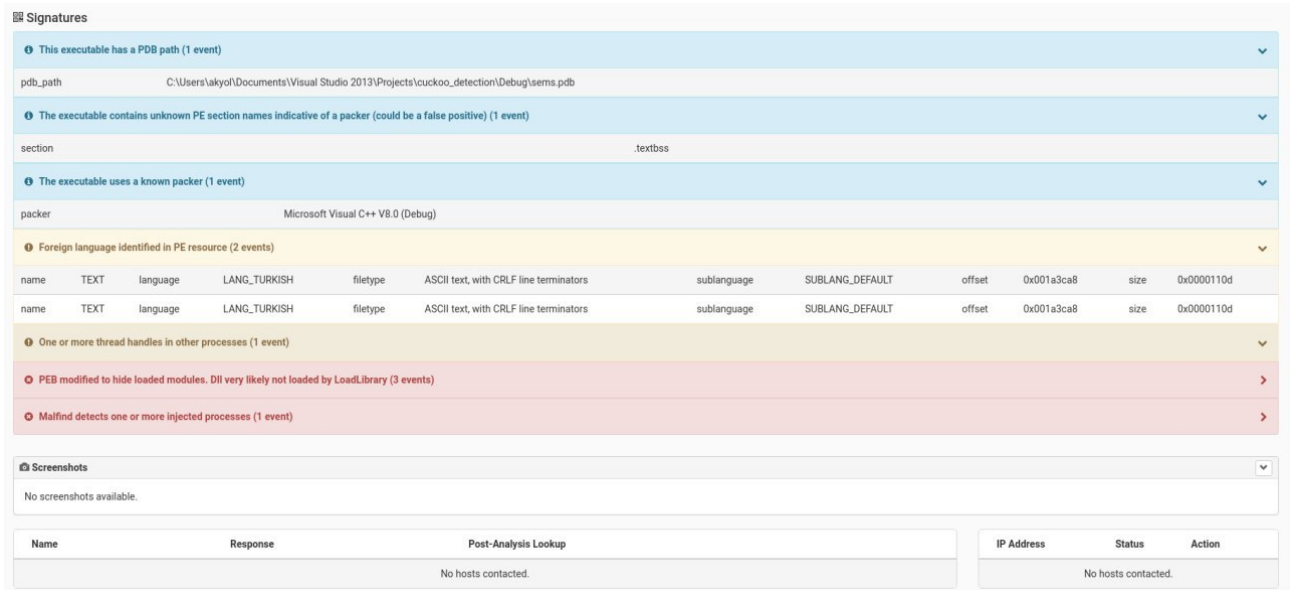

Figure 4: Cuckoo default configuration check

Figure 5: Cuckoo custom configuration check

References

[1] InfoSec Reading Room | SANS Institute

[2] Sandbox | Wikipedia.org

[3] Sandbox explained | cyberbit.com