Shub-Nigurrath [ARTEAM]

Reversing of a Protection Scheme based on drivers: SandBoxie

Version 1.1
Last Rev.: August 2007

Into this Tutorial

Author: Shub-Nigurrath

Forewords

Sometime happens to fall into an interesting protection which reveals to be nicely implemented and nice to describe into a tutorial. This time is the turn of SandBoxie, a program that has an nice protection schema. I thought it could have been useful to reverse and document in a tutorial, mostly because I used a lot a combination of OllyDbg and IDA Debugger.

This time I preferred using IDA as much as possible to understand the code and then OllyDbg only to verify the assumptions done. This method of investigation is usually very common when you have to analyze malware, but also very handy, because IDA allows saving of reversing sessions, code editing, name changing and so on.

I need reversing instruments that could be frozen at any time (I have very few and scattered spare time): I usually run the dynamic sessions with OllyDbg on a VMWARE virtual PC which I can freeze at anytime and the analysis sessions with IDA (which can also be closed and started again later for another session).

I will end this journey doing a complete keygen of the program, showing the process you can use too with other programs and will include in the distribution its sources (simple C).

As usual there are cracks and keygens too for this program around the net and this tutorial will not create many troubles than those already created by someone else before me.

Moreover it will then be the occasion to deeper dig the IDA functionalities in combination with OllyDbg, I will try to be as much clear as possible, for everyone.

Have phun,
Shub

Disclaimers

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this document have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched, and cracked versions were available since a lot of time. ARTeam or the authors of the paper cannot be considered responsible for damages to the companies holding rights on those programs. The scope of this tutorial as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art and generally the comprehension of what happens under the hood. We are not releasing any cracked application. We are what we know..

Verification

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: http://releases.accessroot.com

Table of Contents

This is the tutorial number **200.** We released 200 Tutorials!! This is an astonishing result for a team as our, **200** original tutorials published on our pages! Hip hip hip hurray to us. Long live ARTEAM!!
http://arteam.accessroot.com

1. The Target and its limitations

The target is SandBoxie[1], a program that creates a sandbox for any program running on your system. A sandbox means a protected execution environment which virtualizes the program inserting a virtual layer between the real operative system and the application. This layer will take care of the system calls of the application and will divert them into a safer place, a local database. The target is to have a safer execution place for application without the load of a complete virtual machine like VMWARE. These environments are indeed really popular because offer a safer execution environment without slowing down too much the application, like the real virtualization PC programs (e.g. VMWARE or VirtualPC).

Technically speaking these programs divert the kernel level functions used by the system to access files, registry and memory adding hooks that divert the execution to a local database (where the data come and go). It is the same technique used by the rootkits (see Figure 1).

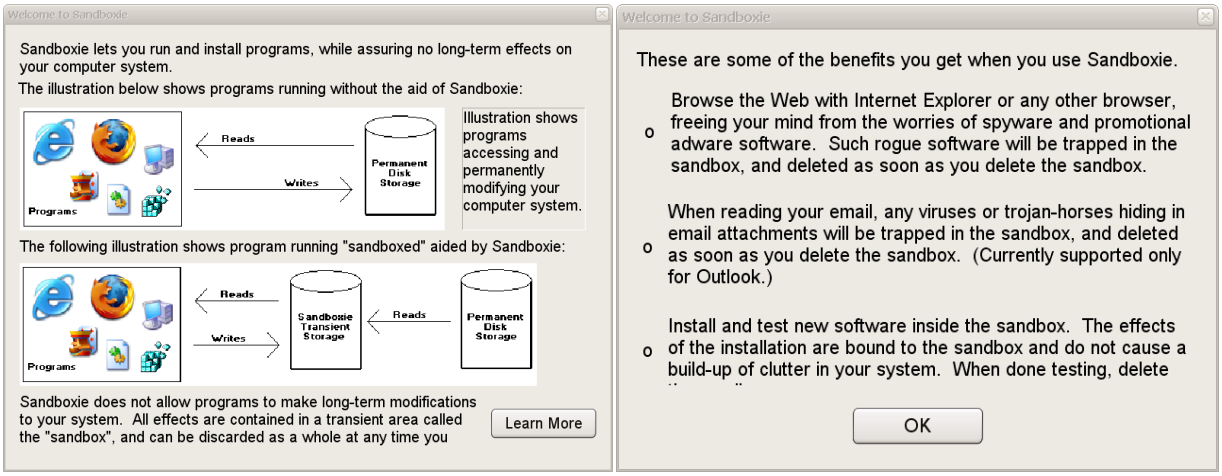


Figure 1 - SandBoxie essentials

Of course the first thing is as usual to install the application and see if and how its components are protected. You can find in the installation directory these executables:

- Control.exe
- SbieDll.dll
- SbieDrv.sys
- Start.exe

There are other executables of course but their names identify them clearly as auxiliary programs, moreover they do not have any graphic resource inside (no dialog, no strings etc).

For all of them the PEiD reports:

	Info
\Control.exe	Microsoft Visual C++ 7.0 Method2 [Debug]
\SandboxieDcomLaunch.exe	Microsoft Visual C++ 7.0 Method2 [Debug]
\SandboxieRpcSs.exe	Microsoft Visual C++ 7.0 Method2 [Debug]
\SbieDll.dll	Microsoft Visual C++ 6.0
\SbieDrv.sys	Microsoft Visual C++ 6.0
\SbieMsg.dll	PE Win32 DLL (0 EntryPoint)
\SbieSvc.exe	Microsoft Visual C++ 7.0 Method2 [Debug]
\Start.exe	Microsoft Visual C++ 6.0

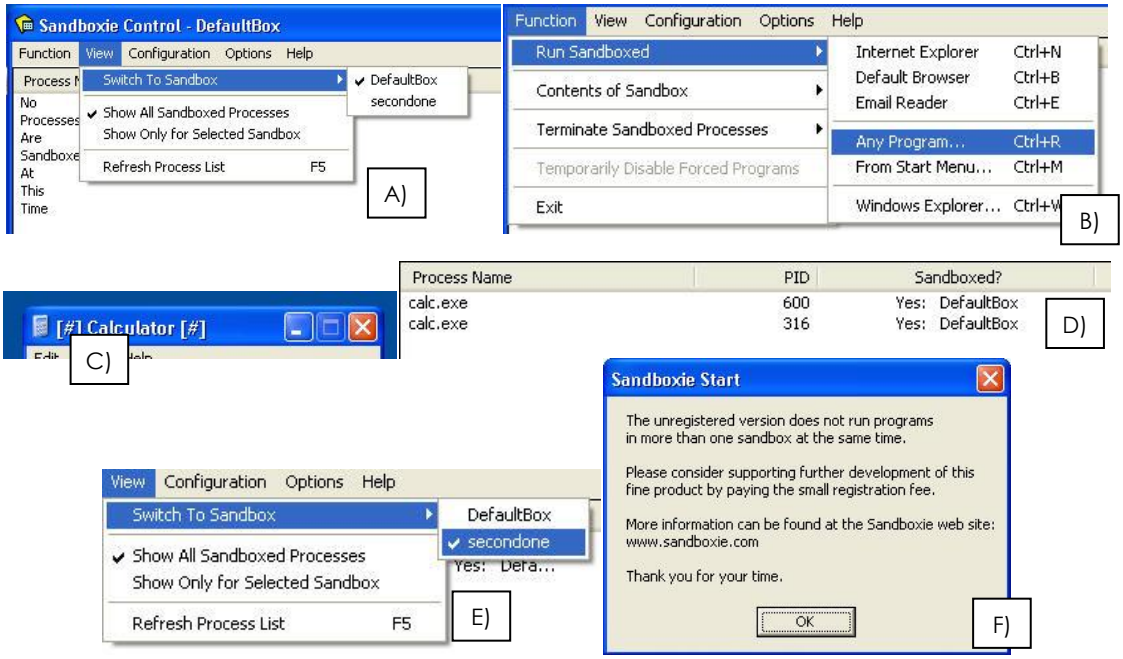
Good, it seems at all a simple target, a target for which a tutorial shouldn't be needed..or not?

1.1. Trial mode limitations

The limitations of the program in unregistered mode are essentially three:

1. 30 day limit of usage
2. Is not possible to run more than one program into different sandboxes
3. No advanced features, like automatic alarms when a program runs out of any sandbox or launching a program always into a sandbox

Try to follow these steps to verify what I told:



A) Create a second sandbox called "secondone" and select the DefaultBox
B) Launch any program into the sandbox, like for example the calc.exe
C) The calc.exe if placed into the sandbox will have a different name with # before and after the window titlebar.
D) The list of processes you can sandbox into a single sandbox is unlimited but try to change the sandbox ..
E) Now select the "secondone" and try to launch again the calc.exe but this time into this second sandbox.
F) You get a message like above..

Other limitations are those in Figure 2, and Figure 4. Figure 3 is interesting also because tells us that the alarm that will be triggered by the system is of type SBOX1118.

Figure 2 – Registration schema and badboy message

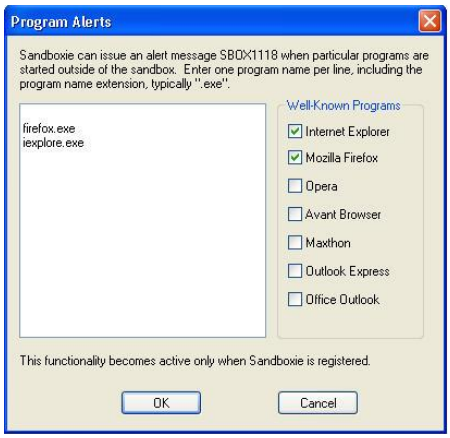


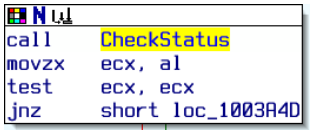
Figure 3 – automatic warning if a program runs outside the sandbox

Figure 4 – about dialog

2. Understanding the protection structure

We know enough of the program in order to start reversing it. We will start from what appears the most logical point: start.exe.

We analyze start.exe using IDA before running it and looking at the start function we can see that the structure is quite simple: it calls some exports of the SbieDll.dll dll and then an interesting function at `01003A35` (call `sub_1003120`). This call contains a lot of interesting messages, among which there are those we saw speaking of limitations. We can then name in IDA this function with a more meaningful name: press N over it and write "`CheckStatus`".



After this call the program takes two directions:

1) Call the function `sub_1002580` with argument 1 (push 1).
2) Go on with the routine.

Option 1) ends calling the `sub_1002580` with parameter 1 pushed on the stack:

```
.text:01003A58                push    1
.text:01003A5A                call    sub_1002580
```

If you go looking at the `sub_1002580` you will see that it's calling `NtTerminateProcess` and that the argument pushed on stack is passed to it as exit code. IDA identified the local argument for you marking it as arg_0 and interestingly if you highlight it with the mouse it also highlights each usage of that name in the following code.

```
.text:010025AD                mov     eax, [ebp+arg_0]
.text:010025B0                push    eax
.text:010025B1                push    0FFFFFFFFh
.text:010025B3                call    ds:NtTerminateProcess
```

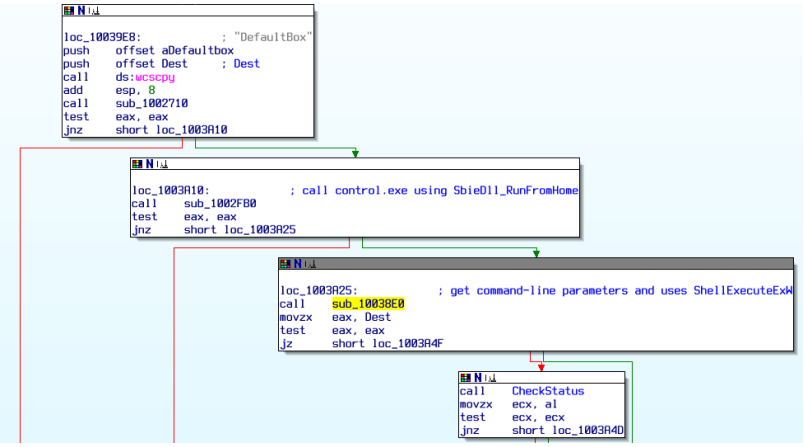
The automatic analysis IDA performs on function is really handy and allows to statically following the parameters of a function.

```
.text:01002580 arg_0           = dword ptr  8
.text:01002580
.text:01002580                 push    ebp
.text:01002581                 mov     ebp, esp
.text:01002583                 push    0FFFFFFFFh
.text:01002585                 push    offset dword_10014C8
.text:0100258A                 push    offset __except_handler3
.text:0100258F                 mov     eax, large fs:0
.text:01002595                 push    eax
.text:01002596                 mov     large fs:0, esp
.text:0100259D                 sub     esp, 8
.text:010025A0                 push    ebx
.text:010025A1                 push    esi
.text:010025A2                 push    edi
.text:010025A3                 mov     [ebp+var_18], esp
.text:010025A6                 mov     [ebp+var_4], 0
.text:010025AD                 mov     eax, [ebp+arg_0]
.text:010025B0                 push    eax
.text:010025B1                 push    0FFFFFFFFh
.text:010025B3                 call    ds:NtTerminateProcess
.text:010025B9                 mov     [ebp+var_4], 0FFFFFFFFh
.text:010025C0                 jmp     short loc_10025DE
```

Note 1: One advantage of IDA is the easiness with which you can rename functions once you understand what are they used for. This allows to better understand the code and immediately recognize a function when it' used. The same functionality is offered by OllyDbg but actually rarely used. Also due to the easiness with which the UDD files get overwritten (each time the original exe is changed the UDD is recreated, except if you have a proper patch to disable this feature).

We can then rename the function `sub_1002580` into a more meaningful "TerminateStart".

The function start.exe call just before the function `CheckStatus` call two other functions:



One is the function `sub_1002FB0`, which is interesting because launches the program control.exe as in Figure 5.

```
loc_1003049:                    ; Size
push    44h
push    0                       ; Val
lea     ecx, [ebp+Dst]
push    ecx                     ; Dst
call    memset
add     esp, 0Ch
mov     [ebp+Dst], 44h
push    10h                     ; Size
push    0                       ; Val
lea     edx, [ebp+var_18]
push    edx                     ; Dst
call    memset
add     esp, 0Ch
lea     eax, [ebp+var_18]
push    eax
lea     ecx, [ebp+Dst]
push    ecx
push    0
push    offset aControl_exe ; "Control.exe"
call    SbieDll_RunFromHome(x,x,x,x)
movzx   edx, al
test    edx, edx
jz      short loc_10030EC
```

Figure 5 – start launches control.exe

The program is passed as argument of the function `SbieDll_RunFromHome(x,x,x,x)` which is a function exported by SbieDll.dll. Start.exe also creates here some mutex to save from multiple instances of the program. IDA fortunately was able to understand that the function `SbieDll_RunFromHome` needs 4 parameters.