© The CodeBreakers-Journal, Vol. 1, No. 1 (2004)
http://www.CodeBreakers-Journal.com

XP SVCHOST Reversed - Services? Processes? How many is too many!

Andrea Geddon

Abstract

This simple essay applies to win xp, I have no service packs installed. I have the enterprise edition. This essay should be similiar for all svchosts of xp. You can also apply this to win 2k but its a bit different, however this is just to give an idea of how to study the problem.

Keywords: *Reverse Code Engineering; XP SVCHOST; Services; Processes*

I. Intro

What is this essay about? Well, since I installed Win Xp I always noticed something strange... Why are there FOUR (not one!) svchost processes? Why are there so many ports open? I feel like I have no control over my pc, and that's no good. So let there be light!

This simple essay applies to win xp, I have no service packs installed. I have the enterprise edition. This essay should be similiar for all svchosts of xp. You can also apply this to win 2k but its a bit different, however this is just to give an idea of how to study the problem.

What else do you need? The TcpView tool is really useful (www.sysinternals.com). I use it but you can substitute it with netstat. Let's start!

II. Processes and Services

Everyone who cares about security will check running processes and open connections on his machine. On win xp i found lots of processes and open services, but I did nothing! I just installed the os! Let's have a look at the running processes.

Here I have all known processes:

- lsass: local security authority subsystem
- csrss: client server runtime subsystem
- smss: session manager subsystem
- winlogon: login/out manager
- system: system process
- idle: idle process
- svchost: see later

and some other processes like explorer, spoolsv etc. Ok nothing strange here, if you want more informations about these components you can read "Inside microsoft windows 2000" book. We should now check our open connections. First thing I do is map services to processes, so i see what processes are responsible for what services. To do this you can use netstat, in the command line type:

```
netstat /ao
 -a option will list also unconnected end points
 -o will show the ID (decimal) of the process owning the service
```

you will see a list of TCP and UDP connections, every one associated to a PID. Now if you download a process viewer which shows you the PID ID's of the running processes. You can check the corresponding process for each connection you see in netstat. If you want an easy tool you can download TcpView, it shows you the connections as you see them in netstat, but they are automatically associated to a process, so you can easily see the relationship between processes and services. Ok, i've done this simple work, and we now see that svchost is involved in a lot of services! In fact, if you double click on svchost in tcpview you will see process properties: "Generic Host process for win32 services". Ok, the svchost is responsible for a lot of services, but why are four instances of the process executed?

Let’s have a look in the code of svchost:

```
.text:01001CE2 push esi
.text:01001CE3 push offset _SvchostUnhandledExceptionFilter@4

.text:01001CE8 call ds:__imp__SetUnhandledExceptionFilter@4
.text:01001CEE push SEM_FAILCRITICALERRORS

.text:01001CF0 call ds:__imp__SetErrorMode@4
.text:01001CF6 call ds:__imp__GetProcessHeap@0

.text:01001CFC push eax
.text:01001CFD call _MemInit@4

.text:01001D02 mov eax, offset _DllList
.text:01001D07 push offset _ListLock
.text:01001D0C mov _DllList2, eax
.text:01001D11 mov _DllList, eax
.text:01001D16 call ds:__imp__InitializeCriticalSection@4
.text:01001D1C call ds:__imp__GetCommandLineW@0
.text:01001D22 push eax
.text:01001D23 call _BuildCommandOptions@4
.text:01001D28 mov esi, eax
.text:01001D2A test esi, esi
.text:01001D2C jz short loc_1001D58
.text:01001D2E push edi
.text:01001D2F push esi
.text:01001D30 call _BuildServiceArray@4
.text:01001D35 call _BuildServiceTable@0
.text:01001D3A mov edi, eax
.text:01001D3C test edi, edi
.text:01001D3E jz short loc_1001D46
.text:01001D40 push esi
.text:01001D41 call _CallPerInstanceInitFunctions@4
.text:01001D46 push esi
.text:01001D47 call _MemFree@4
.text:01001D4C test edi, edi
.text:01001D4E jz short loc_1001D57
.text:01001D50 push edi
.text:01001D51 call ds:__imp__StartServiceCtrlDispatcherW@4
.text:01001D57
.text:01001D57 loc_1001D57:
.text:01001D57 pop edi
.text:01001D58
.text:01001D58 loc_1001D58:
.text:01001D58 push 0
.text:01001D5A call ds:__imp__ExitProcess@4
```

this is the entry point, i’ve pasted all the main body.

I’m not pasting all of the asm source, its too long, i’ll traduce it in C so we can easily follow the operations of this process:

```
CRITICAL_SECTION ListLock;
SERVICE_TABLE_ENTRY SvcTable[];

int __cdecl wmainCRTStartup()
{
    CHAR **CmdLine;

    SetUnhandledExceptionFilter(&SvchostUnhandledExceptionFilter);
    SetErrorMode(SEM_FAILCRITICALERRORS);
    MemInit(GeProcessHeap());
    InitializeCriticalSection(&ListLock);
    CmdLine = BuildCommandOptions(GetCommandLine()); //processes command line
    if(CmdLine != 0)
    {
        BuildServiceArray(CmdLine);
        if((SvcTable = BuildServiceTable()) != 0)
        {
            CallPerInstanceInitFunctions(CmdLine);
        }
        MemFree(CmdLine);

        if(SvcTable != NULL)
        {
            StartServiceCtrlDispatcher(SvcTable);
        }
    }
    ExitProcess(NULL);
}
```

this is an approximate C code for the asm i’ve pasted. Here we see that the command line parameters are fundamental to the functionality of the process, if we run svchost.exe without parameters the process will exit. We can see then that if the command line is set, the program builds first a service array and then a service table. If the service table is correctly built the process connects to the Service Control Manager via StartServiceCtrlDispatcher api. We will focus on the BuildServiceArray and BuildServiceTable sub functions.

```
void BuildServiceArray(char **ParameterString)
{
    PHKEY hKey;

    if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,
                "SOFTWARE\Microsoft\WindowsNT\CurrentVersion\SvcHost",
                NULL, KEY_READ, hKey) == ERROR_SUCCESS)
    {
        ReadPerInstanceRegistryParameters(hKey, ParameterString);
        RegCloseKey(hKey);
        ... (other code)
    }
}
```

I pasted only the code we need, after this there is some code that simply enumerates services, creates the array space, and puts all pointers to services’ strings in that array. So we need to dig into ReadPerInstanceRegistryParameters.

We see the following:

```
.text:01001DAB push ebp
.text:01001DAC mov ebp, esp
.text:01001DAE push ebx
.text:01001DAF push esi
.text:01001DB0 mov esi, [ebp+phkResult]
.text:01001DB3 push offset _ServiceNames
.text:01001DB8 push 7
.text:01001DBA push dword ptr [esi+0Ch]
.text:01001DBD push [ebp+hKey]
.text:01001DC0 call _RegQueryString@16
```

it calls the subfunction RegQueryString. If we go into that function we see:

```
.text:010014D8 push ebp
.text:010014D9 mov ebp, esp
.text:010014DB lea eax, [ebp+arg_C]
.text:010014DE push eax
.text:010014DF push dword ptr [ebp+arg_C]
.text:010014E2 push [ebp+arg_8]
.text:010014E5 push [ebp+arg_4]
.text:010014E8 push [ebp+arg_0]
.text:010014EB call _RegQueryValueWithAlloc@20
.text:010014F0 pop ebp
.text:010014F1 retn 10h
```

that is a wrapper to another function that will probably extract data from the registry. At least i hope so!

```
void RegQueryValueWithAlloc(PHKEY hKey, CHAR* strParam,
                            DWORD sAddr, DWORD param4)
{
    DWORD Size, Type;

    if(RegQueryValueEx(hKey, strParam, NULL,
                       &Type, NULL, &Size) == ERROR_SUCCESS)
    { //this QueryValue simply gets the size of the key
        ...
        if(Size == 0)
        {
            return;
        }
        if(sAddr = MemAlloc(NULL, Size) == NULL) //alloc space
        {
            return;
        }
        RegQueryValueEx(hKey, strParam, NULL, &Type,
                        &sAddr, &Size); //read data
        ...
    }
}
```

This is where the service data is read. The strParam corresponds to a string extracted from the command line and indicates the name of subkey to query data from. Now we can go and search in the registry at

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost
```

key location. We find these keys:

```
imgsvc
LocalService
netsvcs
NetworkService
rpcss
termsvcs
```

each one has a string value, representing the list of services associated to that key. In particular if we check in tcpview we can see from process properties that the svchost processes are launched with the following command line parameters:

```
%SystemRoot%\System32\svchost.exe -k LocalService
%SystemRoot%\System32\svchost.exe -k rpcss
%SystemRoot%\System32\svchost.exe -k netsvcs
%SystemRoot%\System32\svchost.exe -k NetworkService
```

Each process takes care of a different subset of services. Subsets are described in the registry. That's why we have four instances of svchost! Well it was not really necessary to disassemble svchost to understand this, but i love reversing ;-).

If you want you can check the service -¿ process mappings with the command ”sc queryex” in the dosprompt, you will have a list of structures like this:

```
SERVICE_NAME: ...
DISPLAY_NAME: ...
TYPE :        ...
STATE :       ...
WIN32_EXIT_CODE :    ...
SERVICE_EXIT_CODE :  ...
CHECKPOINT :  ...

WAIT_HINT :   ...
PID :         ...
FLAGS :       ...
```

you can look at the PID to see what process owns that service. If you like, you can run services.msc utility to see more information, about services, with a nice gui :-). Now we know more about svchost and services, we can use services.msc to deactivate all unwanted services to close relative ports. I advise you to close all services you do not really need, like UPnP etc. You will not be able to shut down epmap service and microsoft-ds. Dont worry, that's normal. epmap runs on port 135, its the rpc service (remote procedure call, service for DCOM architecture (CORBA for windows)) for distributed applications, if you terminate rpc process a timer will appear that will make you reboot in 45 seconds. Btw, if you need to stop that timer you can use ”shutdown -a” command in dos prompt. However rpc service is vital for the system, so you should not close it. The other open port is for Common Internet File System (port 445), again its a system process and you can't close it. You can use a firewall (or code your own ndis driver!) to filter unwanted extern traffic on those ports. I hope this simple tutorial will clarify the internal management of services. I really hope you will learn to reverse your own binaries to understand the internals of your os :) bye!

III. Greets and Thanks

Thanks to all RET bros! Thanks to Kathras who is helping me very much with another porject at this time. I hope we will release a good version soon! Thanks to Devine9 who always reads and corrects the grammar in my writings!

I hope i can write next tutorial on kernel reversing.

Greets to all UIC members and to all #crack-it people.

GoodBye!

[AndreaGeddon]	andreageddon@hotmail.com	my mail
	www.andreageddon.8m.com	my lame italian site
[RET]	www.reteam.org	RET’s great site
[UIC]	www.quequero.org	italian university of cracking