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

Reverse Engineering the Service Control Manager (SCM)

Doug

Abstract

The SC Manager exported API functions are located in ADVAPI32.DLL (winsvc.h header file in the platform sdk). These functions will ALL end up calling RPCRT4.DLL's NdrClientCall2. If you want to do a lot of tracing, the madness begins there.

Keywords: *Reverse Code Engineering; Service Control Manager; SCM*

I. Intro

There are several examples available on the net that can give src samples on this. ex:

`http://www.codeguru.com/misc/enum_services.shtml`,

Kayaker also put up a nice package of assembly sources. Good Work!

I also have to say that MS' symbolic information loaded in ida/sice proved to be an amazing source of information.

II. My Problem

Main problem I want to know how services (this includes kernel drivers in windows world) are managed internally.

Side-effect problem
Within 50 lines of disassembly, advapi calls `RPCRT4!NdrClientCall2`.

This is essentially debugging the client side of the interface without being able to know what/who the server-side is.

All the way inside RPCRT4!NdrClientCall2, I found an interesting call to `Kernel32!TransactNamedPipe -> \Pipe\svcctl`. I thought I had it all figured out when I saw a `SYSTEM\CurrentControlSet\Services\Npfs\Aliases` string containing "svcctl".

Not quite. `npfs.sys` is just the driver handling Named Pipes (npfs: named pipe file system ?). So I kept on tracing Kernel32's TransactNamedPipe, until I ended up in ring0: `ntfs.sys!NtControlFile` (because afterall a NamedPipe can also be opened with CreateFile). Sounds like I was way off track heh? Thats right, I was.

III. Solution

I ended up coding a wide-char (unicode) file scanner to scan my systemroot dir. It is very primitive scanner, so it can only scan for case-sensitive patterns. No luck searching for all svcctl, SvcCtrl, Svcctl,

But then, I saw in `advapi32!ScWaitForStart`:

```
OpenEventW(100000h,0,<Global\SvcctrlStartEvent_A3752DX>);
```

I rescanned my SYSTEM32 dir for Svcctrl, and it found something in advapi32 (duh) and services.exe (!) I have no idea why I did not notice services.exe before.

A. services.exe

Here's a sample of the non-exported symbols you will find in services.exe

```
ScCreateScManagerHandle(unsigned short *,struct _SC_HANDLE_STRUCT * *)
ScLoadDeviceDriver(struct _SERVICE_RECORD *)
ScQueryServiceStatus(struct _SERVICE_RECORD *,union STATUS_UNION,int)
ScIsValidScManagerOrServiceHandle
ScAccessValidate
ScIsValidServiceName
ScRegOpenKeyExW(struct HKEY__ *,unsigned short *,
                unsigned long,unsigned long,struct HKEY__ * *)
ScRegQueryValueExW(struct HKEY__ *,unsigned short const *,
                   unsigned long *,unsigned long *,unsigned char *,
                   unsigned long *)
ScGetImageFileName(unsigned short *,unsigned short * *)
REnumServicesStatusA(x,x,x,x,x,x,x,x)
ScReadDependencies(struct HKEY__ *,unsigned short * *,unsigned short *)
[...]
```

So the RPC server I was looking for is services.exe:

```
  call    _RpcpInitRpcServer@0 ; RpcpInitRpcServer()
  push    [esp+arg_4]
  push    [esp+4+arg_0]
  call    ?SvcctrlMain@@YGXHQAPAD@Z ; SvcctrlMain(int,char * * const)
  xor     eax, eax
  --
  RpcpStartRpcServer(x,x)
  ...
```

Furthermore, using a tool such as Regmon, it's quite easy to find that a call to an API like EnumServicesA will bring a good set of results.

As for the naming scheme used in services.exe, I have a good feeling procs starting with an 'R' are the ones directly handling the RPC server-side, where as the Sc* are the actual/real functions.
So here's a summary of what goes on when you call an SC Manager API: (ex: EnumServicesStatusA)

```
   advapi32!EnumServicesStatusA
      -> rpcrt4!NdrClientCall2
         -> [...]
         -> rpcrt4!OSF_CCALL::FastSendReceive
            -> rpcrt4!MP_SyncSendRecv
               [...]
               (requested path was found to be a pipe)
               -> kernel32!TransactNamedPipe
                  -> ntdll/ntoskrnl/ntfs/...
                  [...]
```

Link between client and server is not clear, however, it is based on Named Pipes.

```
   -> services!_NdrServerCall2
   [...]
   -> services!REnumServicesStatusA
   [...]
ScFindEnumStart   ; -> This is getting the list to use
                  ;  when looping through ScGetDriversStatus.

*This block is looped through, for each driver.*
*-> services!ScGetDriversStatus
*  -> ntdll!NtQueryDirectoryObject
*   -> ntoskrnl!_NtQueryDirectoryObject ; not exported  accessible
via SYSCALL/int 2E, returns a lot of info
*      [...]
*  (Fill in the blanks in the structure)

Back into the client ....
```

advapi!ScConvertOffsetsW : Fixes the file structure's pointers so that they can be accessed from the user-supplied buffer.

The whole process isn't super-efficient. First, ScFindEnumStart gets a linked list of the ServiceDatabase (each element is a `SERVICE_RECORD` struct). The first link is located inside services.exe data section, and only contains the Forward Link of the list.

Side Note: the ”ServiceDatabase” variable is initialized by ScInitDatabase. ScInitDatabase essentially builds the linked list structure, containing the ServiceType, ServiceName, Forward Link, etc..

Here is the `SERVICE_RECORD` structure with fields that were easy to find.

B. SERVICE_RECORD struct

offset	Name	Comments
0000	PreviousServiceRecord	Back Link in the linked-list
0004	NextServiceRecord	Forward Link in the linked-list
0008	Lp_WideServiceName	Points to Name in wide-char format
000C	Dupe_WideServiceName	Same as Lp_WideServiceName
0010	struct_size	Size of the current structure
0014	**unknown**	
0018	sErv_tag	dd sErv
001C	**unknown**	
0024	Lp_WideFullServicePath	(* see below)
0028	dwServiceType	(** see below)
002C	dwCurrentState	(*** see below)
...		

(*) = Points to full service path\name in wide-char format.
Ex: \Driver\<name> or \Filesystem\<name>
(**) = SERVICE_* fields apply (SERVICE_KERNEL_DRIVER,...)
(***) = SERVICE_* fields apply (SERVICE_RUNNING, SERVICE_STOPPED,...)

After getting the ServiceDatabase linked list, services.exe will loop through each member of the list and for those that are `SERVICE_DRIVER`, it will call ScGetDriversStatus.

Like most of us noticed already, the database that maintains such huge list of services (and not just the ones that are running) is the registry:
`HKLM\SYSTEM\CurrentControlSet\Services`, which in turn has several pointers to other places in the registry.

During this query, it seems the registry is only used to query ’extra’ information, while there is an internal database being actively maintained. ntoskrnl’s NtQueryDirectoryObject does the magic here and returns a list of drivers. services.exe then loops through each name returned by NtQueryDirectoryObject to find the driver for which it received a `SERVICE_DRIVER` structure. Note that NtQueryDirectoryObject may be called more than once for the same `SERVICE_DRIVER`, as it is called using some fixed length buffer and more data may need to be retrieved.

I’m not quite sure why they aren’t calling NtQueryDirectoryObject in the first place, and using its results to build their list, making sure the Service is a driver. The way it is designed, they have two nested loops: the outer loop checks every element from its own ServiceDatabase against NtQueryDirectorys list (inner loop) - Thus calling NtQueryDirectory many-many times; while one loop could have been enough to retrieve the same information.

C. MISC stuff

- Some ADVAPI calls in services.exe will go through the same RPC process, only to come back and finally get handled in services.exe (!)
- To make debugging easier, set a breakpoint where ScGetDriverStatus is called:

```
push    0
push    [ebp+PTR_SERVICE_RECORD]
call    ScGetDriverStatus                ; <-- here

ex: bpx <va> IF(BYTE(*(((EBP->8)->8)))==N) DO D (ebp->8)->8
to break when the ServiceName starts with N
```

D. Possible Ways to find the non-exported functions

1) Implement an IDA-like tracing engine that scans for byte patterns, while ignoring offsets/[...] that easily change between minor revisions. In its simplest form, this isn’t as hard as it may sound to some people.
2) Use Microsoft’s symsrv.dll and dbghelp.dll to get and analyze the symbol in the executable. Use the information provided to find the address of the functions that would be interesting to call directly.

E. Interesting Usage

- Hook the functions that allow to retrieve Services status. Several anti-debugging tricks that make use of the SC Manager have been posted already (ex: Armadillo). Implementing an anti-antidebug util can be interesting for some people.
- Get everything the SC Manager has to offer without being restricted by advapi’s interface.
- Understanding what some obscure registry keys do.

... I’m sure there is more that I can’t think of.

IV. Bonus: anti-SoftICE code + hiding SoftICE

A. Anti-SoftICE

I included a simple program that can detect if SoftICE is active using EnumServicesStatus. Use:

```
EnumServicesStatus_check.exe 0 : to detect softICE
EnumServicesStatus_check.exe 1 : to detect softICE + log all services.
```

The C source code is included.

B. Hiding-SoftICE

There are many many ways to avoid being detected by this simple check. Changing the service name by patching softice + changing registry, doing a case-by-case defense, Kernel freaks could try to hook NtQueryDirectoryObject, I will be playing with services.exe.

Essentially, I will change the status of the driver to `SERVICE_STOPPED`. For this purpose, I will hook ScGetDriverStatus and once it returns, I will analyze the `SERVICE_RECORD` it received as parameter. If the ServiceName is NTice, the dwCurrentState will be overwritten with `SERVICE_STOPPED`.

C / ASM source code is included.

Note that I was a bit sloppy in implementing this hiding util... I did not code ANYTHING that would allow it to resolve symbols into services.exe. Therefore, the address to hook has been hardcoded in the program. Unless you are using the same services.exe build as me, you will have to:

(check comments on top of scDriverStatus.c for specific details)

- get symbols for services.exe
- disassemble services.exe w/ symbols
- seek to the end of the proc ScGetDriverStatus
- find the good virtual address.

For that particular reason, I am not distributing the compiled exe. You *HAVE* to find the value yourself.

Usage:

```
      Launch the program, Install Hook,
      Hook turns active automatically.
      Enable/disable the hook by pressing the middle button.
      Hook is uninstalled when you press Exit
```

It was tested on Windows XP - SP1. Although it was not specifically tested, theres no reason for it not to work on win2K/2003, as long as the symbols for services.exe can be analyzed.

Don’t use this experimental utility if you are doing anything serious with your computer. Im not responsible for problems ;)

V. Conclusion

services.exe has a straight forward interface, it really wasn’t hard to get information from a disassembly. More ”internals” information could be found by digging into ntoskrnl’s NtQueryDirectoryObject, but I’ll leave that for someone else :)

-¿ I still don’t know of a generic/efficient way to debug RPC calls, so far it’s only been a case-by-case analysis.

Please post bugs, problems on RCE Messageboard’s Regroupment (woodmann). Do not e-mail me about this tool.

If someone wants to extend the ScGetDriverStatus hook thing, you have my blessing;)

Well, I hope the information was useful.

doug