

New thoughts in ring3 NT rootkit

author : baiyuanfan

• mail : baiyuanfan@163.com •

August 7, 2005



X'con 2005

Contents

- ◆ The current development situation of NT rootkits
- ◆ New thoughts in ring3 NT rootkit
- ◆ Implement TCP port reuse by hooking asynchronous I/O call
- ◆ Remote control through port of iis6 in ring3
- ◆ Hide myself: self-delete and revive
- ◆ Some other thoughts and discusses
- ◆ Appendix

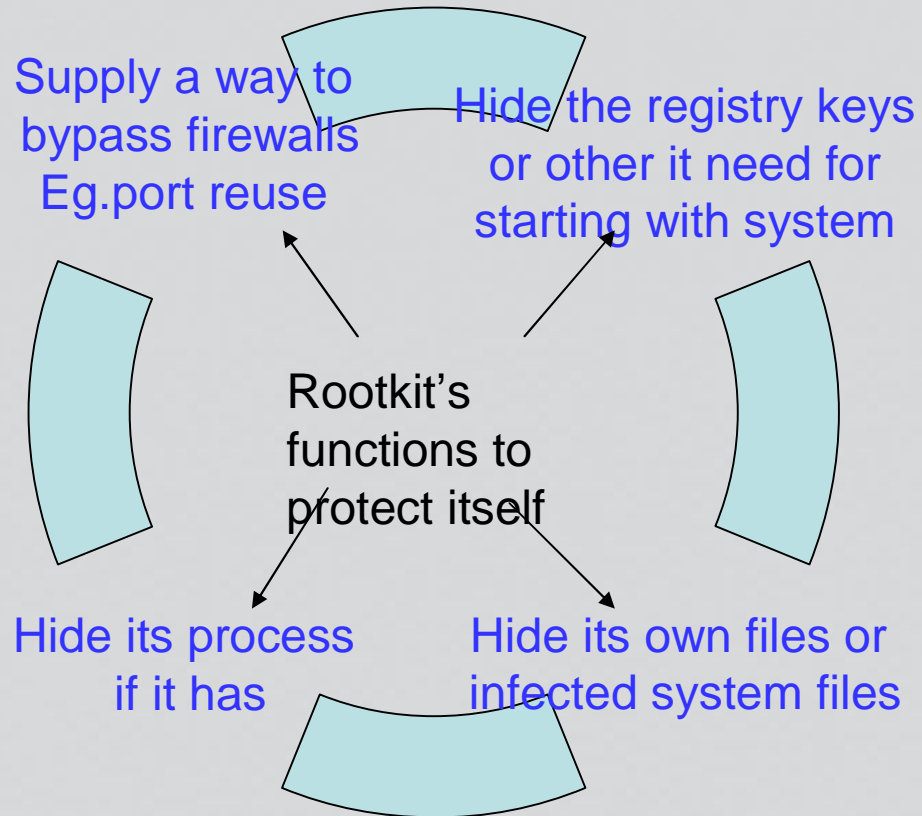


Chapter One The current development situation of NT rootkits

- ❖ As the development of backdoor and anti-backdoor technology, normal sort of backdoors which are more or less just like remote-control softwares became unproper for complicated environments: firewalls, security policies, IDS etc.. Firewalls restrict the applications' opening ports or rebounding connections at will; Various kinds of detectors list the suspectable processes, files and registry keys to detect and kill backdoors. Rootkits were born in this case. They are senior backdoors; They have abilities to hide themselves and bypass the security facilities like firewall.



Chapter One The current development situation of NT rootkits



Chapter One The current development situation of NT rootkits

- ◆ 1. ring3 scope:
 - ◆ Representative product : Hacker Denfender
 - ◆ Hook native APIs in ntdll.dll to hide specific processes, services, files and ports
 - ◆ Effecive upon common checks by administrators; But upon the kernelmode driver level check, it is useless, for the latter check donot go through the hooked executive routine.
 - ◆ Port reuse is implemented through hooking win32 API WSARecv / ReadFile.
 - ◆ Port reuse donot work under new Microsoft IIS6 environment.



Chapter One The current development situation of NT rootkits

- ❖ 2. ring0 scope, many well-known products are just of tech-show kind. In this article I donot discuss rootkit technology in ring0. Briefly, ring0 rootkits are more powerful,more complicated and more noticed by HIDS and anti-rootkit.
- ❖ 3. So we discover that how to design stable,effective port reuse and how to resist anti-rootkit detect from kernelmode are the two important questions to be solved nowadays of ring3 NT rootkit.



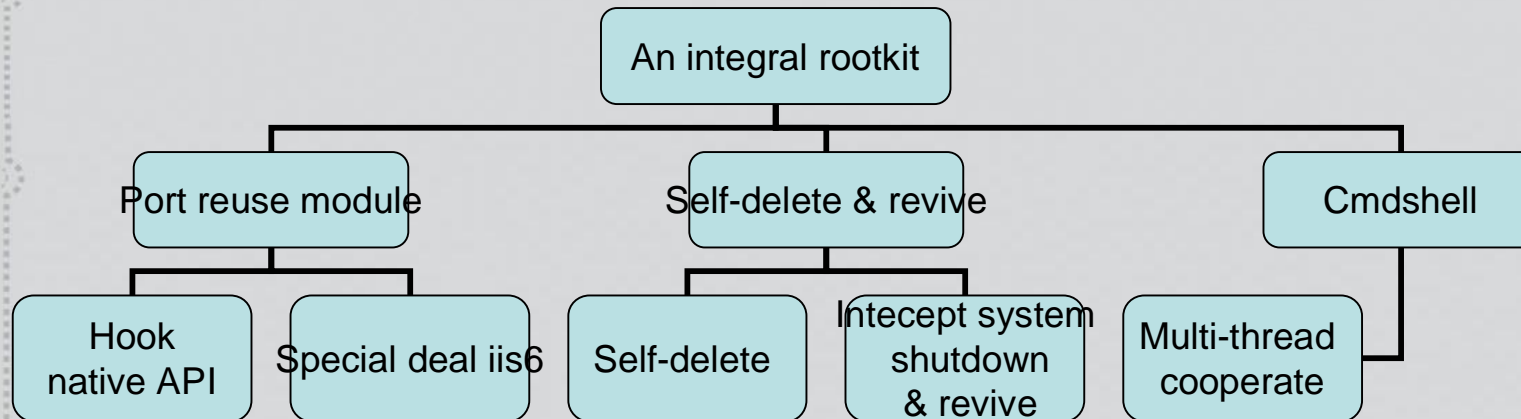
Chapter Two New thoughts in ring3 NT rootkit

- ◆ Integral ideas
- ◆ 1. Implement synchronous and asynchronous port reuse by hooking local system services
- ◆ 2. Specially deal with windows2003's IIS6
- ◆ 3. Thoughts in self-hide: Preferring "NO exist" to "Hide" , resist the backdoor check
- ◆ 4. Multi-threads cooperating system
- ◆ 5. Complete most backdoor work by ourselves, do not work with other backdoors carelessly
- ◆ 6. I implemented an in-test ring3 NT rootkit: byshell v0.67 beta2, to demonstrate my thoughts.



Chapter Two New thoughts in ring3 NT rootkit

- Model of a integral rootkit system

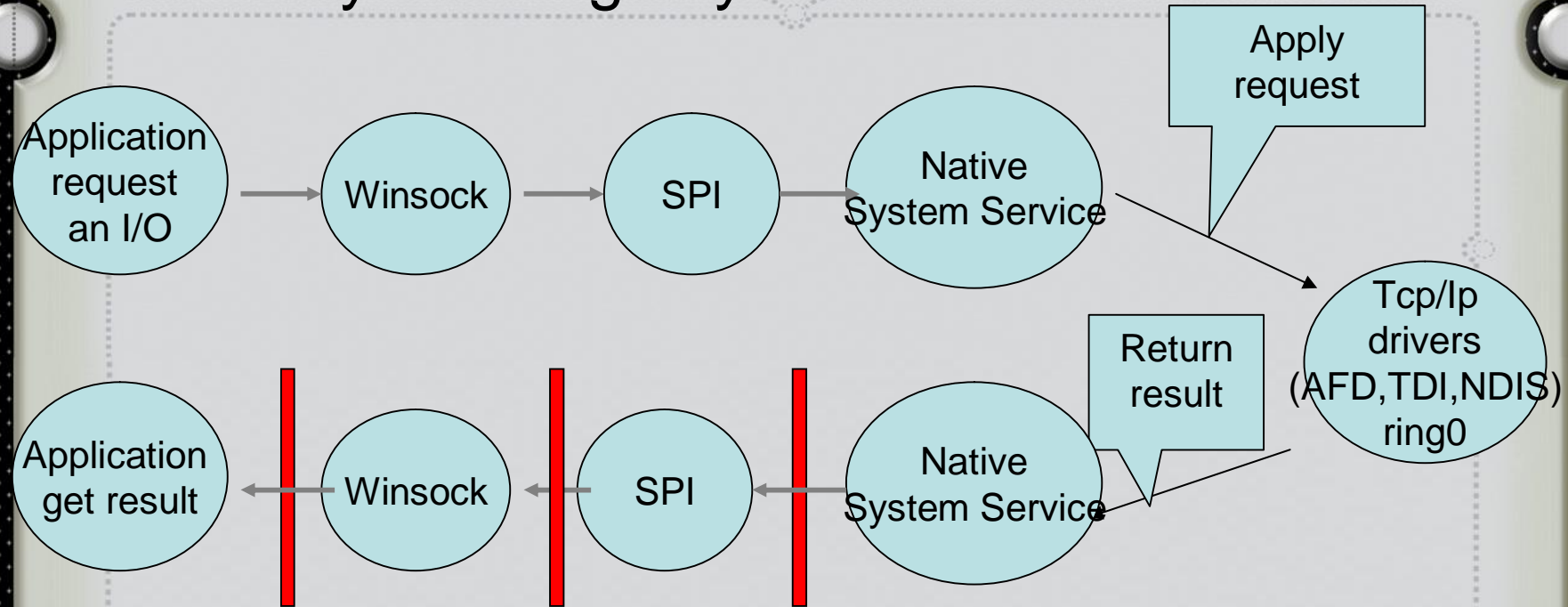


Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ The way to reuse port by hooking network I/O is comparing the received packets before the hooked TCP receive function return to upper caller. If the packet contains specific signal sent by backdoor controller, we grab the socket for backdoor communication and notice the caller that this connection has been closed. So we can implement port reuse that donnot affect normal service already bound on that port.



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call



Here we have several chances to intercept the result in ring3, so we just do `if(!strcmp(buff,"signal"))(dobackdoor());`



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ Hooking synchronous network I/O is simple that we can directly compare the data after the real function returned to us and before we return to caller.
- ◆ But towards asynchronous I/O call, there are two difficulties :
 - ◆ we cannot directly compare the data after the real function returned to us and before we return to caller, since when the real function returned to us, the data in buffer was not updated. Later when the data updating completed, the system will notice the caller through other methods.
 - ◆ winsock library offer too many kinds of I/O models in win32 level. This could confuse us and let we not know how to hook.



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

1. Backgrounds: native API level I/O module

- ◆ Through hooking native API level network I/O functions, we can get rid of the mass of win32 level I/O models.
- ◆ As I know all the windows native APIs dealing with I/O give its caller 3 choices. Synchronous I/O, asynchronous I/O that notice the operate completion by event object, and asynchronous I/O that notice the completion by user APC routine.

- ◆ We can have a see on ZwReadFile, which invoked by win32 level well-known function ReadFile.

- ◆ NTSYSAPI
- ◆ NTSTATUS
- ◆ NTAPI
- ◆ ZwReadFile(
IN HANDLE FileHandle,
IN HANDLE Event OPTIONAL,
IN PIO_APC_ROUTINE ApcRoutine OPTIONAL,
IN PVOID ApcContext OPTIONAL,
OUT PIO_STATUS_BLOCK IoStatusBlock,
OUT PVOID Buffer,
IN ULONG Length,
IN PLARGE_INTEGER ByteOffset OPTIONAL,
IN PULONG Key OPTIONAL
);



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ If caller set up an APC routine, the system ignores the event object parameter, deal this I/O as asynchronous I/O that notice the completion by user APC routine, and will callback its APC routine when the I/O was complete. If caller donot give an APC routine but do give an event object, it will be dealed as asynchronous I/O that notice the completion by event object, when the I/O was complete the system will signal the event object. If caller set up neither, system deal it as synchronous I/O and will block the caller until the operate was complete.
- ◆ no matter how complex we feel the various I/O models in win32 level, like such "completion port" model, they just depend on these 3 basic methods. Now we can just process these three and need not to discuss various complex win32 level I/O models. So we solved the sencond difficulty refered above.

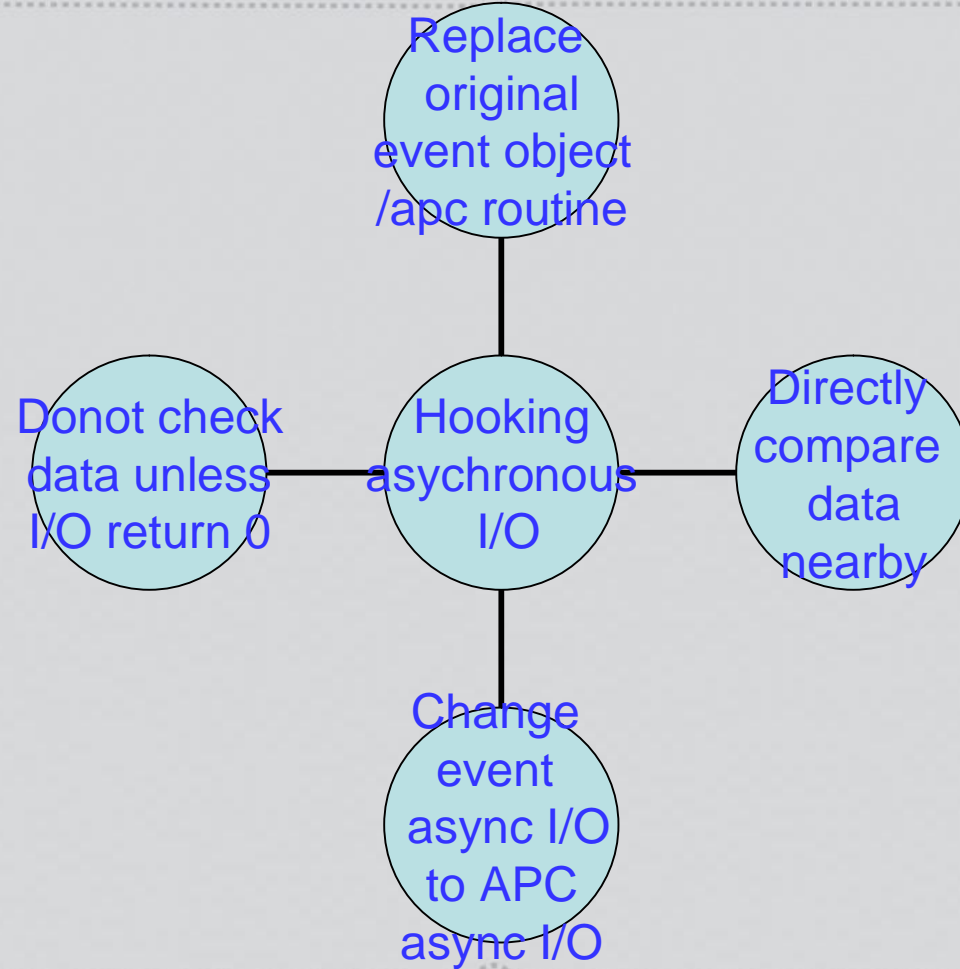


Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ❖ 2. Four methods of hooking native level asynchronous I/O
 - ❖ Now we discuss the most troublesome first difficulty. Towards asynchronous I/O how and when could we compare the data in buffer? There are four methods to be select.



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ A. Replace the event object or APC supplied by caller to ours
 - ◆ The way to replace APC is easy. When a call using APC complete-notice comes we change the APC routine parameter to ours and save original APC routine and context in our APC context. When I/O completed our APC routine was called by system. Here we just compare the data in buffer, if it matches the signal of controller we execute backdoor, else we transmit the APC to caller through our saved original APC routine and context.
 - ◆ Towards event complete-notice asynchronous I/O, we cannot set up a single event to replace the caller's event and transmit the signal. Since when many concurrent requests comes, we cannot know which original event is corresponding and should be transmitted to. We can set up many new event objects and matches them to original events one by one, but it can obviously decrease the efficiency of whole system .



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ❖ B. Donot check data unless I/O return 0
 - ❖ If the packets from client arrive at the server's network driver faster than the applications on server invoke a receive function, the I/O may return success(0) instead of pending(0x103) .
 - ❖ This method need backdoor client's cooperate to send a lot of same packets in a short time. The hooked function will not check the data in buffer unless the asynchronous call directly returns success. If the call directly returns success, the data in buffer was updated already.
 - ❖ But unfortunately, experiments proved this method has a low success rate, especially towards IIS.



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ C. Directly compare the data in buffer
 - ◆ No matter which returned status is, pending or success, directly search the buffer or nearby memory for the signal .
 - ◆ this method will not only cause error decisions but also cause missing signals sometimes.
 - ◆ But we can design a part of error controlling code to control the error or missing in a tolerable extent.
 - ◆ After we used error controlling code as above, some missing of signals still can happen. I have designed an network application model to bypass the directly comparing method. Current network server applications almost never use such way so the rootkits' purpose of reusing port can always succeed.
 - ◆ My demo rootkit used this method.



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ◆ D. At the end of editing of this article I've got a new idea to improve method A.
- ◆ The original method A can manage APC complete-notice async I/O but cannot manage event complete-notice async I/O. We can change the caller's event complete-notice async I/O to our APC complete-notice async I/O. We can delete the caller's parameter event object, fill the APC routine parameter with our APC routine. And we save original event object in our APC context. Then we can execute the function.
- ◆ When our APC routine is called, the I/O operation has complete. We just compare the data in buffer with our specific signal. If not matching, we can transmit the complete-notice to original caller by `SetEvent(APCContext->eventobject);`



Chapter Three Implement TCP port reuse by hooking asynchronous I/O call

- ❖ As above, this could be an excellent solution without any error or missing, and also has a high efficiency. This can be used not only in port reusing of rootkit, but also in cases that need a high level of precision like packet sniffers or loggers.
- ❖ In the limit of time left, this new method lacks enough test . I cannot confirm whether or not there will be some side effect . For a rootkit that donot demand precision, a simple method C is enough.

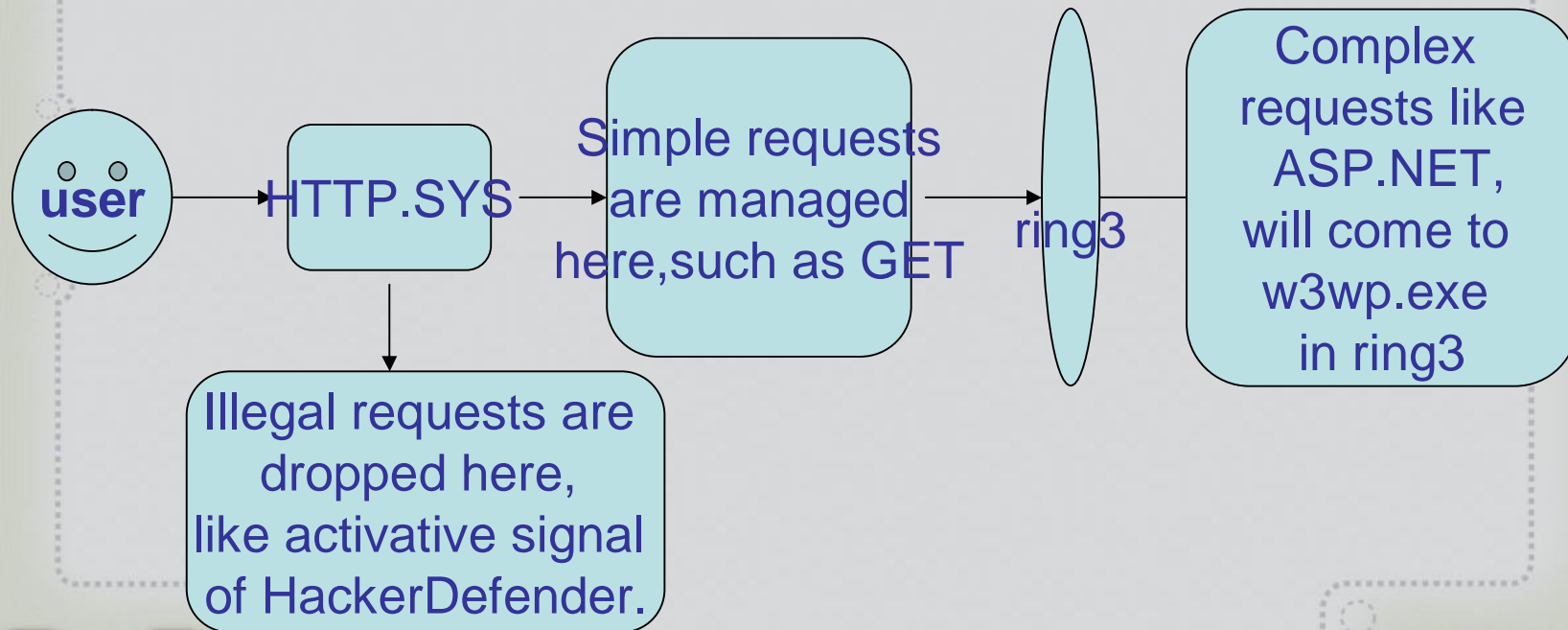


Chapter Four Remote control through port of iis6 in ring3

- ❖ Now IIS6 has been popular for winnt based network server. Rootkits like Hacker Denfender is unable to control its TCP port, for the connections in IIS6 is managed by a device driver http.sys which is unable to be hooked by ring3 code.
- ❖ Then we need a procedure transferring our commands in form of legal requests from ring0 to ring3 and intercepting it in ring3 .



Chapter Four Remote control through port of iis6 in ring3



Chapter Four Remote control through port of iis6 in ring3

- ◆ 1. Basic knowledge: IIS6's security mechanism
 - ◆ Connections in IIS6 is managed by a device driver, not traditional ring3 application. The driver will interpret the user's request first. If it consider the request illegal or simple as GET a file, it can manage it itself.
 - ◆ If the request is complex like asp.net, it will transfer the request to a ring3 low-privileged process w3wp.exe. The latter deal with it and returns result to driver to send to user.
 - ◆ we cannot grab the socket as we want. Even we could intercept the signal and execute the command from controller, we must find another method to transfer the command's result to controller.



Chapter Four Remote control through port of iis6 in ring3

2. Solution

- ◆ Use a complex protocol like soap to pass the command through the kernel level to user level. Then we can brutally search the stack memory of w3wp.exe to find the command.
- ◆ If we put the command in the soap target filename, we can find it in stack. So we can construct such soap target filename:
 - ◆ `"/abc/12345678baiyuanfangff"//backdoor` signal that matches http and xml syntax
 - ◆ `"1324"//repeating commands distinguishing code`
 - ◆ `"ABCABCABCABC"//encoded command`
 - ◆ `".asmx"//soap suffix`
- ◆ The max length of soap filename is about 256 characters. This is enough for execute commands, but I think it not proper for transferring files, for a soap request need a long time to be finished.



Chapter Four Remote control through port of iis6 in ring3

- ❖ 3. How to send the result to controller
 - ❖ We should write the result in a specific temp file in web directory. The controller could send a simple GET request to fetch the result. Going on like this, we implement an equivalent "connection".
 - ❖ We must know the web directory to write temp file in it. The web directory of IIS5 can be read from:
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Virtual Roots\ /`
 - ❖ But for IIS6 this path become invalid.



Chapter Four Remote control through port of iis6 in ring3

- ◆ Interpret metabase.xml to get every virtual web directory of IIS6 and decide a default one
 - ◆ There is such information in config file of IIS6, metabase.xml:

```
<IIsWebVirtualDir      Location
="/LM/W3SVC/1/ROOT" AccessFlags="AccessRead |
AccessScript" AppFriendlyName="default application"
AppIsolated="2" AppPoolId="DefaultAppPool"
AppRoot="/LM/W3SVC/1/ROOT" Path="g:\www" />
```
 - ◆ The 1 in the string "/LM/W3SVC/1/ROOT" can be any number. And the number represent different virtual web sites. We just need to find the minimum one, which represents the default web site.



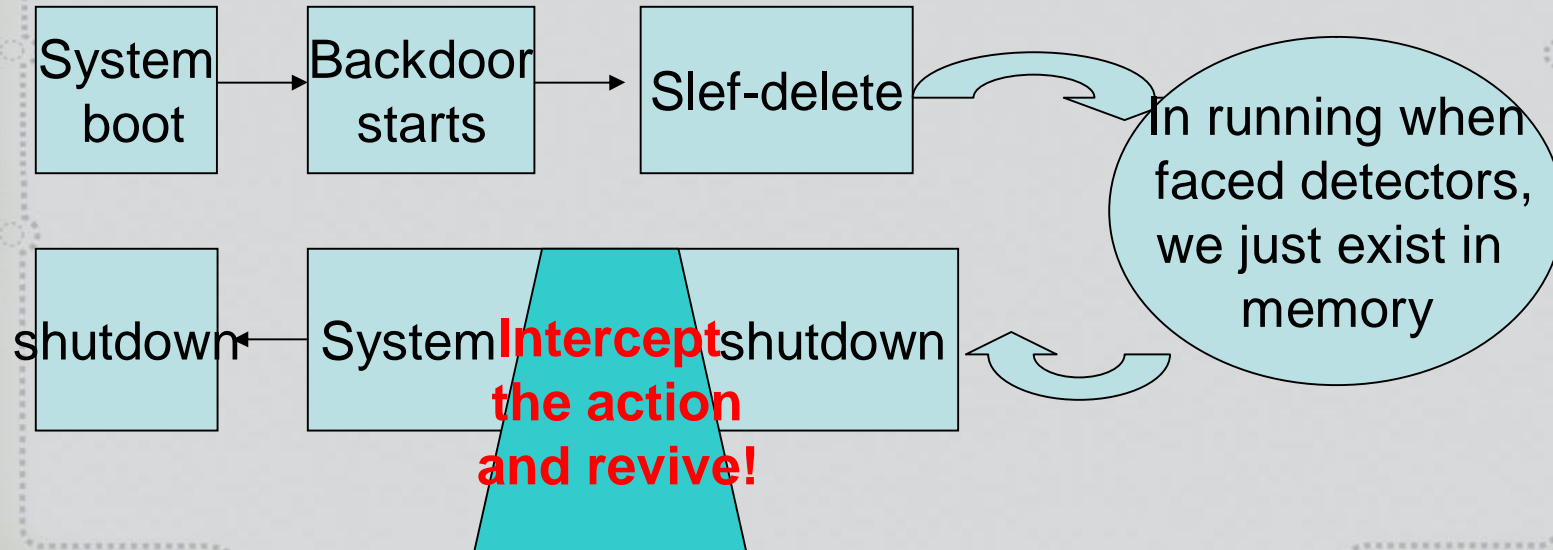
Chapter Five Hide myself:self-delete and revive

- ❖ Preferring "NO exist" to "Hide"
- ❖ the backdoor just exist in memory in form of threads or hooks, and delete all the files and registry keys to let detectors from usermode or kernelmode discover nothing. When the system is about to shut down, we just spawn our files and registry keys.
- ❖ If anti-rootkit starts earlier than us and begins to monitor every action, we can be detected.



Chapter Five Hide myself:self-delete and revive

◆ Preferring "NO exist" to "Hide"



Chapter Five Hide myself:self-delete and revive

- ◆ A similar thought started by nongmin in his cmdbind2
 - ◆ His main idea is to cooperate with a dll-injection backdoor. The injector first inject a small function to remote process. The function load the dll in remote process' address space. Next save the dll memory at another place, unload the dll. Then allocate a block of memory with the same address to the previous loaded dll, copy the saved dll memory to this address. Finally the dll can continue working, but the dll disappear in the process' loaded dll list. So he could delete all the files, when system shutting down he spawn his file.
 - ◆ His method is not proper for a rootkit. We need many changes.



Chapter Five Hide myself:self-delete and revive

- ◆ His method just suits inject dll to one process onetime. Rootkit must infect all processes and manipulate newly created process in shortest time.
- ◆ In order to infect new process immediately after it was created, we must hook function about process creation. Hooking ZwCreateprocess(Ex) is ineffective, for at that time only a process object has been created, many works concerned have not been completed. Functions like ReadProcessMemory cannot work properly at that time.
- ◆ So we need to hook ZwResumeThread. In this function we should decide whether the process is a new one and do hooking if it is.



Chapter Five Hide myself:self-delete and revive

- ❖ Original dll has been deleted at time of new process' creation, We cannot use LoadLibrary easily to load dll to target process.
 - ❖ We can manually relocate all the dll codes. Manually interpreting .reloc section and relocate all the things is possible. But this cost a lot of time.
 - ❖ We can imitate Hacker Defender, which donot use dll, all its codes are self-relocatable written by assembly. But if we use assembly to write all codes self-reloactable, the work will be too much to implement a big backdoor function module.



Chapter Five Hide myself:self-delete and revive

- ❖ We can just copy all the dll image to the new process. This demands the dll-base address has not been occupied or there will be an error. We can use an uncommon address as our dll's base address.
- ❖ Another question is that these kernel32 functions' addresses we used have not been reloaded. If the new process and its parent process have different kernel32 baseaddress, there will be a crash.
- ❖ But this case just can seldom happen in some system-supported process like csrss.exe, these system-supported processes donot take part in the work of new process creation.



Chapter Five Hide myself:self-delete and revive

- ❖ Intercepting the signal of shutting down
 - ❖ A common method is to use `SetConsoleCtrlHandler()` to register a `CTRL_SHUTDOWN_EVENT` dealing routine in a `WIN32_OWN_PROCESS` type service process to receive shutdown signal.
 - ❖ Finding a proper and NT version independent service process is not so easy. `Spoolsv.exe` maybe a in-comparison good choice . But some systems that donot use printer could have banned it.
 - ❖ Also we cannot choose a system-supported process like `winlogon.exe`. Maybe there are something specially in such processes, they can never receive any signal said above.



Chapter Five Hide myself:self-delete and revive

- ❖ Intercepting the signal of shutting down
 - ❖ We should adopt hooking the function which should be called when shutting down the system. There are two of such functions to choose: kernel32!ExitWindowsEx and ntdll!ZwShutdownSystem.
 - ❖ Since some virus or such kind of applications use the latter to fast shutdown, obviously the latter is more reliable.
 - ❖ But unfortunately, when the latter is called, some part of subsystem seems to be unavailable, we cannot write files and registry keys successfully. So we can only hook kernel32!ExitWindowsEx to revive when shutting down.



Chapter Six Some other thoughts and discusses

- ◆ Following are some my unripe ideas, just for discussion.
- ◆ 1. Anti-ring3rootkit method 1: monitor the creation of remote thread
- ◆ 2. Anti-ring3rootkit method 2: hook ZwWriteVirtualMemory
- ◆ 3. Anti-ring3rootkit method 3: check if important APIs have been hooked by import/export
- ◆ 4. Is there any new idea on anti-ring3rootkit?
- ◆ 5. When the ideas about anti-ring3rootkit above have been implemented, how could ring3 rootkits exist any longer? Are we only have one single way to step into ring0?



Appendix:

◆ byshell v0.67 beta2

- ◆ I implemented an in-test ring3 NT rootkit: byshell v0.67 beta2, to demonstrate my thoughts. This rootkit has been tested successfully in my own computer under windows2000 (sp4), XP (sp2) and 2003 (sp0).
- ◆ Welcome to all for testing or improving it. Its source is full opened.



Appendix:

References:

- ◆ [1]Hacker Defender
- ◆ Holy_Father(holy_father@phreaker.net)
- ◆ <http://rootkit.host.sk/>

- ◆ [2]cmdbind2
- ◆ nongmin(nongmin.cn@yeah.net)
- ◆ <http://nongmin-cn.8u8.com>

- ◆ [3]ADE32
- ◆ Z0mbie
- ◆ <http://z0mbie.host.sk/>



Appendix:



Thanks:

Thanks to friends in CVC(www.retcvc.com) and xfocus(www.xfocus.net) for your great help! Especially thanks to vxk in CVC. Wish all friends happiness.



About author:

- ◆ Yuanfang Bai, student of 2004 of Software Engineering Academic, East China Normal University, interest in system/ kernel develop and security, like making friends with everyone.

◆ Contact me: [E-mail: baiyuanfan@163.com](mailto:baiyuanfan@163.com)

Thanks!

