# Global Information Assurance Certification Paper

**Managing vulnerabilities exposed by Windows services**
**The Butler did it…in my office…with an open window.**

**GSEC Practical Requirements (v1.4b) (August 2002)**

**James R Williams**
**August 28, 2003**

## 1.0 Abstract

The butler did it…in my office…with an open window. This "Murder Mystery" analogy may seem a strange way to view the vulnerabilities of Windows services. Nonetheless, the analogy provides a very simple way for us to look at the Windows business class operating system, its "services" in particular. In order to fully grasp the items of discussion we need to be able to understand how the Windows operating system works. To do this we need to further our analogy by taking a look at the suspects and the location of our murder mystery. First we have the Windows operating system, a house if you will, "The House of Windows". The work we do in this house, that is to say the applications we use, and the documents we create we will call our "office". All of the methods used to access the operating system will shall consider "windows". While doors may seem a more appropriate analogy, you will rarely find a house with more than two or three doors, but most houses contain several windows of varying shapes and sizes, much like the Windows services. Finally let's refer to the operating system functionalities as servants. These are the individuals that make "The House of Windows" usable. For instance, we can consider the operating system shell, in the case of Windows "Windows Explorer", the nanny. The nanny over seas everything we do, and applies restrictions on our behavior. The file system we will refer to as the housekeeper, it helps us organize, maintain, and categorize all that we deal with. Most importantly we will refer to Windows services as the butler. In the case of our murder mystery analogy, the butler is the force keeping the house that is Windows functioning. The office where you work, in this case the Windows operating system itself, is the location where the crime will be committed. Finally the open window, the vulnerability or exposure, is what lets a person or program with malicious intent into "The House of Windows" to commit the crime. Our friend the butler, the Windows Service, is a necessary and helpful part of the modern day personal computer. However, if you don't keep a close watch on this butler, he'll leave the 'window" open when it is raining. Until recently the Windows service has been a stable and functional part of the Windows business class operating system. Vulnerabilities in the Windows services have gone either undetected, or have just been overlooked in lieu of larger vulnerabilities. With the advent of the "W32.Blaster.Worm" virus in August of 2003, the ability to manipulate Windows Services has come into the spot-light as a method for creating havoc and masking malicious activity.

The intent of this document is to identify the vulnerabilities exposed by the Windows services, and to learn how to secure "The House of Windows" from

these vulnerabilities. For the purposes of our discussion we will be primarily focusing on the Windows XP Professional and Windows 2000 operating systems. As the newest of the Microsoft "business" class operating systems, Windows XP and Windows 2000 have the widest variety of tools, and available settings to manipulate a service into an exposure. That is not to say that much of what we discuss will not apply to older Microsoft operating systems as well (Windows NT 4.0), as much of what we will discuss is backwards compatible, and some historical perspective is required. To fully examine the impacts generated by vulnerabilities in a Windows service we will be taking a look at what makes up a Windows service, the risks posed by the Windows services, how these risks occur, tools for manipulating Windows services, and finally we will review the solutions and resolutions to these vulnerabilities.

## 2.0 Windows Services – Definition.

Returning to our analogy of "The House of Windows", let's take a look at who the butler is and why we hired him. In other words, "What is a Windows service?" The Windows service is the foundation on which Microsoft built its flagship business operating system. The Windows services are used to maintain the operating system functionality as well as to control the extremely sensitive areas of the operating system. Controlled by the Windows "kernel", or brain, the Windows services are a collection of applications used to manage the operating system both during the boot-up cycle, as well as during use. Many of the basic Windows services control how the operating system interacts with other entities. In past operating systems (Windows 3.1) and in many of the "home" PC operating systems (Windows 95', Windows 98', and Windows Me) Microsoft relied on a loose collection of virtual device drivers and dynamic link libraries (DLL's) to manage functionality.[1] In "home" PC operating systems and in older versions of Windows there was also no inherent security. All files were executed or run under the context of the user. Failure of these applications and functions would often result in a system "crash", or a reboot of the operating system to restart the program. With the advent of the Windows "business" class operating system and the use of Windows services, it became possible to give over a degree of control to the operating system and allow these features to run under an "elevated" right known as "System". Under the current Windows "business" class operating systems there are four levels of privilege. The "user" privilege grants access to run applications and use the basic functionality of the system. "Power user" privileges are granted to those users who require more control over the system but do not need completely unrestricted access. "Administrator" privileges are reserved for those individuals who will be responsible for installing hardware, software, and administrating the system. The final level of access, System or "Local System" is reserved for services and operating system functions that run at the highest level of access to administrate those areas of the operating system protected under normal use. Running the service under "LocalSystem" privileges separated the activity of the user from the operating

[1] Microsoft Press, Microsoft Windows 98 Resource Kit

system. This separation along with the advent of 32-bit applications stabilized the program running as a Windows service, and helped to create a faster more crash resilient operating system. It is very important to understand how changing the control of operating system functions from the "user" context to an elevated right affects the operating system. This change is the key in some cases to the vulnerabilities we will be discussing throughout this document. Looking at an example we can review the "spooler" service, or officially the "Print Spooler" service. This service controls how the Windows operating system interacts with printing tasks and devices. A small executable, using the core operating system dynamic link libraries, this service maintains control of all printer related activities and data going to and from the printer. As a core feature to the operating system the Print Spooler service runs with "LocalSystem" privileges, having access to not only the physical ports of the PC where hardware devices may be connected, but also to the network based "virtual" ports used for network printing. That is how a simple feature of the operating system, came to possess the highest level of privileges available to the operating system and still maintain access to all physical and virtual ports on the PC. The spooler service is just one example of the many imbedded services in the Microsoft Windows operating system.[2] Curiously Windows imbedded services are not the only type of service. Many applications, particularly on the server platform, are installed as a service. This is done so that the application can run under the control of the operating system without requiring a user to log in to the system. Many other applications are written as services simply to take advantage of the "service manager" which governs the services for the operating system and ensures stability of the application.

### 2.1 Windows Services – Risks and Vulnerabilities.

Now that we've addressed what a Windows service is, let's move on to the risks that are posed by Windows services. Going back to our "Murder Mystery" analogy "Did the butler leave the window open?" The realistic truth to that question is yes, the window is open. There are two types of "open window" that must be addressed when discussing the riskd associated with Windows services. First and probably most visible are failures in individual services which allow them to be manipulated by a malicious program. As we discussed in the previous section of the document a service is nothing more than a program that is run in a segregated and controlled environment. Unfortunately, most of those programs run in the "controlled" environment are granted the enhanced privilege of "LocalSystem". This means these applications when vulnerable grant an extremely high level of access to anyone who knows how to take advantage of defects or "bugs" in the program. An interesting example of this type of vulnerability was the "W32.Blaster.Worm" virus released in August of 2003. The virus was written and designed to exploit the Windows RPC (remote procedure call) service.[3] This specific vulnerability of this service allowed requests sent to

[2] Glossary of Windows 2000 Services, Web Site
[3] Microsoft Security Bulletin MS03-026, Web Site.

the RPC service on port 135 to run code on the system with "LocalSystem" privileges. The part of this virus that seemed to cause the most damage was the unintended effect of repeated requests on port 135. The service, running under "LocalSystem" privileges is by default set to restart the system on failure. As the virus spread, systems which were being broadcasted with TCP/IP packets on port 135 were constantly rebooted while they remained on the network. While this was most likely not the intention of the virus author, the chaos created by this "side effect" helped to slow down properly patching systems and efforts to clean up the virus. The second type of "open window" related to the Window services are malicious services themselves. As yet an unexploited area of Windows services, creating and installing your own service, remains a viable risk. Installing an application, which runs in the background as part of the operating system with elevated "LocalSystem" privileges, leaves a malicious programmer the perfect sleeper virus. It is possible with very little knowledge, and a couple of tools from the Windows 2000 resource kit to configure and install a service to run any application you choose with "LocalSystem" rights.

### 2.2 Windows Services – Vulnerabilities: How and Why They Exist.

The discussion of risks posed by Windows service vulnerabilities brings us to the next logical step in fully understanding these exploitations. How were these vulnerabilities created and why do they exist? In other words, "Why would the butler leave this many "windows" open?" The answer to this question is as diverse as the number of "windows" within the operating system. Simply put the first kind of Windows service vulnerability we discussed is the result of a defect in an application's code, or an unforeseen use of a feature in the application. Since the birth of the Microsoft "business" operating system, Windows services have been plagued with vulnerabilities and the resulting need for patches (hot-fixes). For instance, take the 1999 vulnerability exposed in our friend the Print Spooler service.[4] This vulnerability took advantage of "open buffer's", unrestricted memory spaces, to allow an application to be run under the "LocalSystem" credentials. API's (Application Programming Interfaces) used by the Print Spooler had no restrictions on the memory space they could use. Therefore, a random string of data would crash the service. However, the vulnerability existed in that a properly manufactured string of data could be used to run an application on the remote system with the credentials of the service, in this case "LocalSystem" rights. In another case the "RunAs" service in Windows 2000 was shown to be vulnerable to a denial of service type attack in November of 2001.[5] This vulnerability was low risk as "administrator" rights were required to perform the denial of service attack required to shutdown the service. This does however, serve as another example of how vulnerabilities can be found and exploited within an existing Windows service. How does this happen? Simple, developers are not perfect, and no test group can be as diverse or critical as the consumer market. Even with proper testing and security hardening procedures

[4] Microsoft Security Bulletin (MS99-047), Web Site.
[5] Denial of Service Vulnerability in Windows 2000 RunAs Service, Web Site

in place, mistakes happen. The real issue relating to this type of Windows service vulnerability is how often and how fast patches can be delivered to the systems, which we will address later in this document. The other type of the Windows service vulnerability is a different issue all together. With the stability and functionality built into the Windows service, it is little wonder that administrators and developers would want to take advantage of this feature. Added to the stability of the Windows service, is the ability to use the elevated "LocalSystem" rights to allow "user" level individuals the ability to perform specific functions on the operating system. Microsoft allows us to take advantage of the Windows service through a series of resource kit utilities. We will discuss these utilities in greater detail in the following section. For now using a Windows service can, for a security administrator, be a life saver. Take for example the case of the Windows operating system command line tool "nbtstat.exe". This tool is used to obtain information about a systems network connection, NetBIOS names, and WINS cache. A vulnerability was found with the "NetBT" service which allowed NetBIOS address spoofing to occur on a Windows network using WINS (Windows Internet Naming Service). This vulnerability is an example of a defect in an application as we discussed earlier.[6] However, the resultant patch for this vulnerability, through an inadvertent change to a shared DLL (dynamic link library), restricted the use of nbtstat to only those individuals with "administrator" level access. On a large network where client (workstation) level security access restriction is in place (everyone is not an administrator) it becomes a problem for most users in the course of troubleshooting problems to execute the nbtstat command line tool. Through the use of Windows services, it is possible to put a GUI (graphical user interface) application on the system for the individual to use. This GUI when run as a service with "LocalSystem" privileges allows the individual to perform the functionality of "nbtstat" without exposing the vulnerabilities of the full "nbtstat" tool. While this is an example of how we use a Windows service to assist in administration and support of the Windows operating system, it is easily turned into a vulnerability. By simply replacing my GUI with an application that records IP addresses or passwords, we can change a helpful operating system tool into a malicious application. What's more, with a service running under "LocalSystem" credentials I have the ability to disable all "administrator" level access to the system. This would prevent anyone from stopping or uninstalling the service. That is assuming anyone noticed the service running on the system as it is running silent in the background of Windows. With the preceding examples we can see how vulnerabilities are created in a Windows service, both through defective code, as well as self written malicious services.

### 2.3 Windows Services – Service Tools.

Now that we have a better understanding of the risks posed by Windows services let's take a detailed look at some of the tools available for manipulating Windows services. To return to our "Murder Mystery" theme, "How did the all of these

[6] Microsoft Security Bulletin (MS00-047), Web Site

"windows" get opened without anyone noticing?" How do you manipulate Windows services? All of the methods available for manipulating Windows services are too numerous to address in this document. What we will discuss here is the primary methods used by non-developers to manipulate the Windows Service. Most imbedded applications (those included with the Microsoft operating system) or application designed to run as a service are built that way at design time. As we've discussed earlier running applications as a Windows service can have tremendous benefits. Additionally, any developer with the knowledge of an object oriented programming language, like C++, can simply design his application from the get go to run as a service. However, if you're only development experiences consists of Visual Basic, if you have no development experience, or if you do not possess the source code of your application there are still methods for running your application as a service. Microsoft provides a resource kit tool called "srvany.exe". This simple resource kit utility can be used to run any executable as a service, provided you follow all the steps necessary.[7] Simply put the tool, when the service is installed, handles the request from the service manager and executes the application specified in a registry entry created for the tool. This "tricks" Windows into believing the code that it is running was in fact written and designed to run as a service. To use the "srvany" tool, you need to install the "srvany.exe" file on the workstation and install a service (refer to the use of Instsrv.exe detailed below), of your choosing, pointing to the srvany executable. This tool has gone relatively unnoticed by system administrators because you must create and configure a registry key to run your program as a service. Knowing where to create this key and what type of information to publish in the key is not documented very clearly in any of the Windows resource kits.[8] "Srvany.exe" is a wonderful tool, and can be a great help in administering a Windows environment. However, by default the service runs with "LocalSystem" privileges and is non-interactive (hidden from the user). This means that services created with the "srvany" tool are left relatively unmonitored. The risk associated with this tool, is that the registry key storing the path and file name of the executable to be run as a service can be edited by anyone with privileges to the registry key. This would allow that person to run any designated application with "LocalSystem" privileges. Tools such as "instsrv.exe" and "sc.exe" are available from the resource kit as well. These tools are used extensively to install, delete, and manipulate the settings of a service. To install a service using the "instsrv.exe" utility and the "srvany.exe" utility the following command line would be run:

```
Instsrv  MyServiceName C:\winnt\system32\srvany.exe
```

The "MyServiceName" entry above represents the name of my service as it would appears in the registry hive with the other services. The path "C:\winnt\system32\srvany.exe" is the path to the location of the "Srvany" tool. Additionally the "Srvany" tool can be renamed and stored with the associated

[7] Minasi, (This old resource kit.) Srvany, Web Site.
[8] Minasi, (This old resource kit.) More about Srvany, Web Site

application it is launching as a service. The "sc" tool can be used to further manipulate the service in altering any number of parameters. Microsoft has provided us with an excellent document on their web site (Retrieving Service Properties) detailing the "service properties" and how they interact with the Windows GUI for editing services. One extremely valuable use of the "sc" tool is the "config" function which allows you to switch the "interactive" property of a service allowing it to interact with the users desktop. This technique can prove most useful when capturing keystrokes from an unsuspecting user. Another tool used to manipulate the Windows services is a resource kit utility called "subinacl.exe". The "subinacl" tool is traditionally used to set file permissions on Windows files and folders. However, among the options available to the "subinacl" tool is the ability to set permissions on a Windows service. Using such a tool, it is possible to revoke the access of the "administrator" to perform tasks on the service. This includes starting and stopping the service, as well as uninstalling the service. Furthermore the "subinacl" utility also possesses the ability to set permissions on registry keys as well. This would allow the installer, of the service, to manipulate access to the registry keys storing the information required to run the service. Fortunately for the security administrator the "subinacl" tool can be used to his advantage. Using its registry features, an administrator can restrict access to the registry keys which store the information the service manager uses to launch the service. The "subinacl" tool can be put to excellent use by restricting the "start/stop" privileges of a service from those users who would not need to interact with the service in this manner.

### 3.0 Conclusion

Having looked at what comprises a Windows service, the risks involved with Windows service vulnerabilities and how these vulnerabilities work, and finally having reviewed some of the major tools used to manipulate the Windows services, we come to the next and last step in addressing these vulnerabilities. Using our "Murder Mystery" analogy we must decide what to do with our friendly butler. What resolutions or solutions are available to address Windows service vulnerabilities? As with all methods of securing vulnerabilities, no single solution is available. We've looked at several examples of defective code which, through no intentional fault, have created several vulnerabilities having some costly risks. While better coding methods, and more extensive testing may address some of the issues we have reviewed, these would still not address exploited features of a service that were never intended be used in a malicious way. Furthermore, that would still not have addressed the use of the Windows service to execute malicious code either. The only solution to these types of vulnerability is to layer multiple methods of protection together. This multi-layered solution while still not exploit free would provide the best means to protecting systems for Windows service exploits while allowing the systems to remain functional for the user. The best method to attempt to manage Windows service vulnerabilities would have to come in a phased approach. First we must secure the "user", then we must secure the operating system, and finally we must secure the services

themselves. Let's begin by taking a look at the first phase of the solution: "Securing the user". One of the main issues facing the Windows "business" class operating system is that many users have more rights then they need to perform their job duties. In many cases, this is accepted as the "norm". Bringing about change is always a difficult task. Microsoft has made our task more difficult in that to install hardware on a Windows XP system requires administrative privileges. However, simply restricting users to only logging in with "administrative" rights when necessary drastically reduces the risk associated with an individual allowing Windows service vulnerabilities to exploited. The "VBS_Loveletter" virus, arguably one of the most costly viruses in the last 5 years, still had to be executed by a user. Had the VBS_Loveletter virus installed its own hidden service the ramifications could have been even worse. Restricting a user's access level prevents this type of "accident" from becoming a "business" tragedy. Reducing the risk of the individual user executing malicious code with "administrator" rights is paramount in reducing the number of exploits available with Windows services. Having secured the "user" we should now move on to the next phase of our solution: "Securing the operating system". At the time this document was written, the W32.Blaster.Worm virus and its variants are still listed as a "severe" risk.[9] This virus and its variants take advantage of an RPC service vulnerability documented by Microsoft on July 16th 2003. Almost a full month before the first version of the "W32.Blaster.Worm" virus was released. Since it is difficult to test code, especially Windows services, for every possible outcome and vulnerability, it is imperative that we patch the exploits we know about as soon as possible. With the release of the Windows 2000 and Windows XP operating systems many new methods of software distribution are available. In cases of high risk vulnerabilities, distribution methods such as Windows login scripts and group policy startup scripts are fast and accurate methods for deploying patches. While testing and deploying hot-fixes and security patches is time consuming and difficult, it is important to take a proactive initiative for preventing Windows service exploits. With the rapid growth of high speed internet access the availability and capacity to exploit vulnerabilities is growing. As we've seen with the "W32.Blaster.Worm, the time between vulnerability assessment and virus delivery is rapidly shrinking. In the past it was possible to deploy vulnerability solutions on a reactive basis, today that is no longer an option. We've defined phases one and two of our solution to securing Windows services. Let's move on to the final phase: "Securing the Service". Securing Windows services requires a change in focus. Currently most software installs that use a Windows service are not reviewed. This means that if a service executes under the "LocalSystem" credential, but only needs run with the user's credentials, the system administrator is unaware. Many developers will allow the service to run under the default security privilege ("LocalSystem"), simply out of convenience. By utilizing the "subinacl" resource kit tool, we can prevent the average "user" from having access to manipulate this type of service. Most users do not require any privileges greater than Start/Stop to use an application. Where imbedded Windows services are concerned most users require no

[9] Symantec, Latest Virus Threats, Web Site

access. Imbedded services are, generally speaking, either “automatic” (launched at boot-up time or log in) or “dependent” (started by another service). When dealing with services written and installed by the system administrator, it is important to closely review how the general “user” will interact with the service. It is also important, when using the “srvany” resource kit, to secure the registry key for the service that stores the program path to be executed. In an environment which is constantly undergoing development and which can pose high level of risk with exposed vulnerabilities, the Windows service must be secured. We know what a Windows service is, the risks involved, how and why Windows service exploits come to exist, some tools for managing Windows services, and the appropriate resolution for securing windows services. Looking at our “Murder Mystery” analogy, “What should we do with our friend the butler?” We can’t fire him, get rid of the Windows service, because we need him to run “The House of Windows”. However, we still cannot leave him working unsupervised. The best solution then is to close the “windows”, and only let the butler open one, when we allow him to.

**References:**


1.) Microsoft Press. Microsoft Windows 98' Resource Kit. Redmond: Microsoft Press, 1998. 494

2.) Microsoft. "Glossary of Windows 2000 Services". 31 July 2001. URL: http://www.microsoft.com/windows2000/techinfo/howitworks/management/w2kservices.asp (28 August 2003)

3.) Microsoft. "Buffer Overrun In RPC Interface Could Allow Code Execution (823980)". Microsoft Security Bulletin MS03-026. 25 August 2003. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS03-026.asp (28 August 2003)

4.) Microsoft. "Patch Available for "Malformed Spooler Request" Vulnerability". Microsoft Security Program: Microsoft Security Bulletin (MS99-047). 04 November 1999. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS99-047.asp (28 August 2003)

5.) SecuriTeam. "Denial of Service Vulnerability in Windows 2000 RunAs Service". 15 November 2001 URL: http://www.securiteam.com/windowsntfocus/6K00F1535I.html (28 August 2003)

6.) Microsoft. "Patch Available for 'NetBIOS Name Server Protocol Spoofing' Vulnerability". Microsoft Security Bulletin (MS00-047). 27 July 2000. URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS00-047.asp (28 August 2003)

7.) Minasi, Mark. "(This old resource kit.) Srvany". February 2000 URL: ***http://www.winnetmag.com/Articles/Index.cfm?ArticleID=7959*** (28 August 2003)

8.) Minasi, Mark. "(This old resource kit.) More about Srvany". March 2000 URL: ***http://www.winnetmag.com/Articles/Index.cfm?ArticleID=7959*** (28 August 2003)

9.) Knowles, Douglas. "W32.Blaster.Worm". 28 August 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html