**Attack Surface Analyzer Readme**

**In This Document**

Overview

System Requirements

Instructions

Collecting attack surface information with .NET Framework 4 installed

Collecting attack surface information without.NET Framework 4 installed

Analyzing the Results

FAQ

Known Issues

Test Results: Security Issues

Executables with Weak ACLs

Directories Containing Objects with Weak ACLs

Registry Keys with Weak ACLs

Processes with Weak ACLs

Process Threads with Weak ACLs

Processes with NX Disabled

Services Vulnerable to Tampering

Services with Fast Restarts

Vulnerable Named Pipes

Vulnerable COM Classes

Vulnerable DCOM Classes

Memory Mapped Sections with Weak ACLs

Test Results: Attack Surface

System Information

Running Processes

Executable Memory Pages

Windows

Kernel Objects

Impersonation Tokens

Windows Stations

Desktops

Modules

Service Information

Services

Drivers

ActiveX, DCOM, COM, File Extensions

Registered COM Controls

Registered ActiveX Controls

Registered DCOM Servers

DCOM Default Permissions

File Registrations

Internet Explorer

Pluggable Protocol Handlers

Silent Elevation Entries

Silent Elevation Entries (RunDLL32)

Preapproved Controls

Browser Helper Objects

IE Zones

IE Actions Policy

Network Information

Ports

Named Pipes

RPC Endpoints

Network Shares

Network Interfaces

Firewall

Firewall Rules

Firewall Profiles

Firewall Service Restriction Rules

Authorized Applications

System Environment, Users, Groups

%PATH% Entries

Accounts

Groups

Group Membership

Account Privileges

Additional Information

Contact

Other Resources

**Overview**

Attack Surface Analyzer is developed by the Microsoft Security Engineering Center (MSEC). It is the same tool used by Microsoft's internal product groups to catalogue changes made to the operating system attack surface by the installation of new software.

Attack Surface Analyzer takes a snapshot of your system state before and after the installation of product(s) and displays the changes to a number of key elements of the Windows attack surface.

This allows the following:

* Developers to view changes in the attack surface resulting from the introduction of their code on to the Windows platform
* IT Professionals to assess the aggregate attack surface change by the installation of an organization's line of business applications
* IT Security Auditors to evaluate the risk of a particular piece of software installed on the Windows platform during threat risk reviews
* IT Security Incident Responders to gain a better understanding of the state of a systems security during investigations (if a baseline scan was taken of the system during the deployment phase)

**System requirements**

**Supported operating systems:**

* Windows Vista
* Windows 7
* Windows 8
* Windows Server 2008
* Windows Server 2008 R2
* Windows Server 2012
* Windows Server 2008 Server Core installation option
* Windows Server 2008 R2 Server Core installation option
* Windows Server 2012 Server Core installation option

**Collection of Attack Surface data:**

* Windows Vista
* Windows 7
* Windows 8
* Windows Server 2008
* Windows Server 2008 R2
* Windows Server 2012
* Windows Server 2008 Server Core installation option
* Windows Server 2008 R2 Server Core installation option
* Windows Server 2012 Server core installation option

**Analysis of Attack Surface data and report generation:**

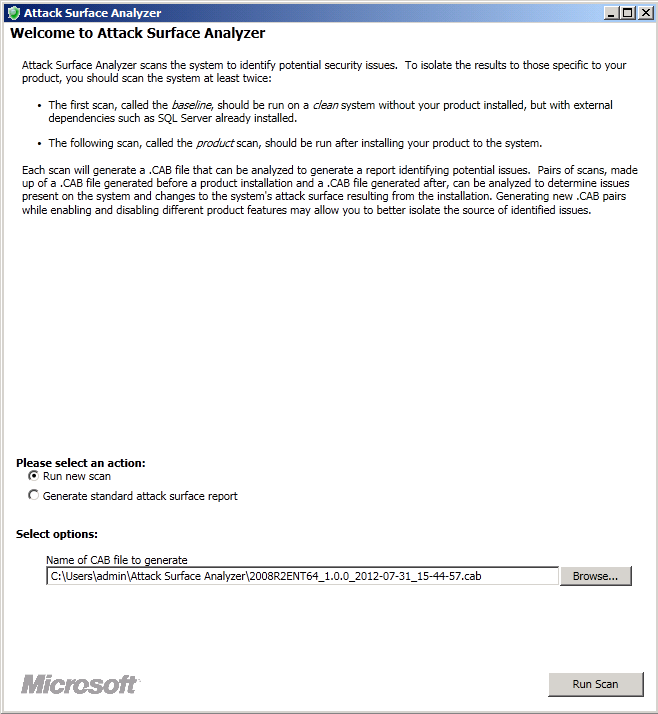
* .NET Framework 4 is required for all analysis and report generation
* Windows 7
* Windows 8
* Windows Server 2008 R2
* Windows Server 2012
* Windows Server 2012 Server Core installation option

**Instructions**

To run Attack Surface Analyzer, you will require Administrative privileges on the computer.

**Collecting attack surface information** **with the .NET Framework 4 installed**

1. Download and install Attack Surface Analyzer on a machine with a freshly installed version of a supported operating system, as listed in the System Requirements section. Attack Surface Analyzer works best with a clean (freshly built) system. Not running the Attack Surface Analyzer on a freshly built system requires more time to perform scanning and analysis.
2. Install any software prerequisite packages before the installation of your application.
3. Run Attack Surface Analyzer from the Start menu or command-line. If Attack Surface Analyzer is launched from a non-elevated process, UAC will prompt you that Attack Surface Analyzer needs to elevate to Administrative privileges.



1. When the Attack Surface Analyzer window is displayed, ensure the "Run new scan" action is selected, confirm the directory and filename you would like the Attack Surface data saved to and click Run Scan.
2. Attack Surface Analyzer then takes a snapshot of your system state and stores this information in a Microsoft Cabinet (CAB) file. This scan is known as your baseline scan.
3. Install your product(s), enabling as many options as possible and being sure to include options that you perceive may increase the attack surface of the machine. Examples include; if your product can install a Windows Service, includes the option to enable access through the Windows Firewall or install drivers.
4. Run your application.
5. Repeat steps C3 through C5. This scan will be known as your product scan.

**Collecting attack surface information** **without the .NET Framework 4 installed**

**Note:** The (command line) method is recommended when .NET Framework 4 is not installed. To perform analysis and report generation, a machine with .Net Framework 4 is required.

1. Download and install Attack Surface Analyzer on a machine with a freshly installed version of a supported operating system, as listed in the System Requirements section. Attack Surface Analyzer works best with a clean (freshly built) system. Not running the Attack Surface Analyzer on a freshly built system requires more time to perform scanning and analysis.
2. If your Windows installation does not have the .NET Framework 4 installed, you have an option of updating your .NET Framework installation or installing only ASA.exe and dependent components. If you choose to install .Net Framework 4, please see the above section, “Collecting attack surface information with the .Net Framework 4 installed”.
3. Navigate to the Attack Surface Analyzer installation directory. The default installation directory is C:\Program Files\Attack Surface Analyzer\.
4. Run Attack Surface Analyzer.exe from the command line. If Attack Surface Analyzer.exe is launched from a non-elevated process, UAC will prompt you that Attack Surface Analyzer needs to elevate to Administrative privileges. To view the full list of command line options execute the command:

““Attack Surface Analyzer.exe” /?” (without the surrounding quotation marks) from the console.

1. Attack Surface Analyzer will then take a snapshot of your system state and store this information in a CAB file, saving the results to your user profile directory - the default is: C:\Users\%username%\Attack Surface Analyzer\. This scan is known as your baseline scan.
2. Install your product(s), enabling as many options as possible and being sure to include options that you perceive may increase the attack surface of the machine. Examples include; if your product can install a Windows Service, includes the option to enable access through the Windows Firewall or install drivers.
3. Run your application.
4. Repeat steps C3 and C5, this scan will be known as your product scan.

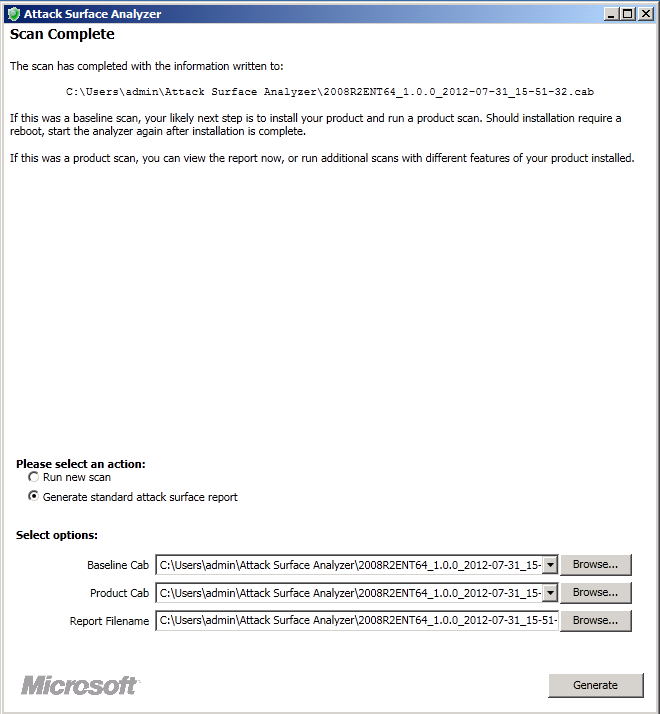
**Analyzing the Results**

You can either analyze the results on the computer you generated your scans from, or copy the CAB files to another computer for analysis. To perform analysis and report generation, a machine with .Net Framework 4 is required:

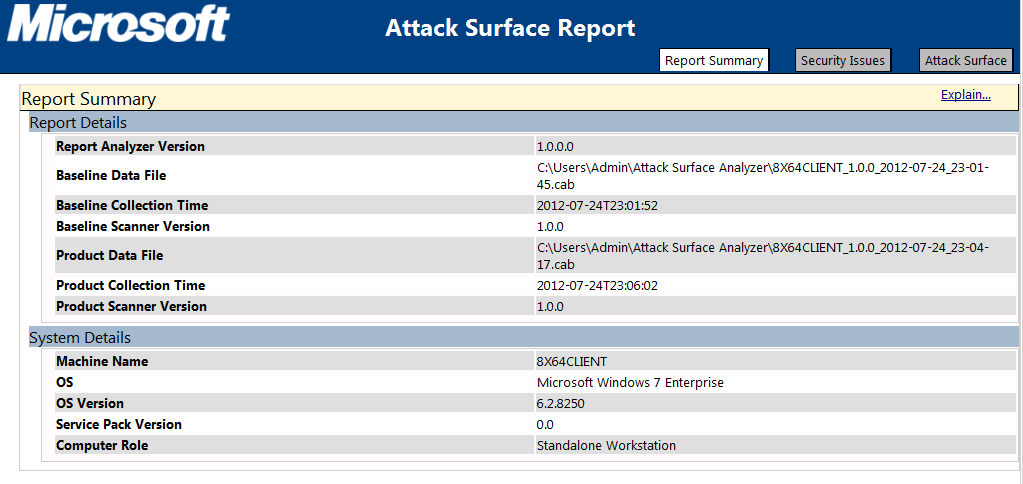
1. Run Attack Surface Analyzer from the Start menu. If Attack Surface Analyzer is launched from a non-elevated process, UAC will prompt you that Attack Surface Analyzer needs to elevate to Administrative privileges.

**Note:** To view the full list of command line options, including generating the report from the command line, execute the command: ““Attack Surface Analyzer.exe” /?” (without the surrounding quotation marks) from the console.

1. Choose the "Generate Report" action and specify your baseline and product scan CAB files. **Note:** Make sure that you have the cab files selected for both baseline and product correctly, then generate report.



Attack Surface Analyzer will inspect the contents of these files to identify changes in system state and if applicable important security issues that should be investigated. If a web browser is installed on the machine performing the analysis it should automatically load Attack Surface Analyzer's report - it is a HTML file.

1. Review the report to ensure the changes are the minimum required for your product to function and are consistent with your threat model.

After addressing issues generated from the tool you should repeat the scanning process on a clean installation of Windows (that is, without the artifacts of your previous installation) and re-analyze the results. As you may need to repeat the process a number of times, we recommend using a virtual machine with "undo disks", differencing disks or the ability to revert to a prior virtual machine snapshot/configuration to perform your attack surface assessments.

**FAQ**

**Is Attack Surface Analyzer V1 compatible with the Beta version of the tool?**

Attack Surface Analyzer V1 is not compatible with the beta version. New scans are required to generate the baseline and product scans. If you are currently using the beta version in a development cycle, we recommend that you keep a copy of the beta version until this current cycle is complete. You can download the beta version of the tool for a limited time here: [x64](http://download.microsoft.com/downloads/1/4/f/14fe12d0-7d89-45de-bd98-ef497d66be5d/Attack_Surface_Analyzer_BETA_x64.msi) and [x86](http://download.microsoft.com/downloads/1/4/f/14fe12d0-7d89-45de-bd98-ef497d66be5d/Attack_Surface_Analyzer_BETA_x86.msi).

**How do you display all of the command line options?**

To view the full list of command line options execute the command:

““Attack Surface Analyzer.exe” /?” (without the surrounding quotation marks) from the console.

**How do you install Attack Surface Analyzer on a machine without .NET Framework 4?**

When installing on a machine without .NET Framework 4 installed, an advisory message is displayed, continue with installation. Run ASA.exe using the command line options.

**Does Attack Surface Analyzer support ARM?**

Attack Surface Analyzer Version V1, does not currently support ARM.

**Which version of “Attack Surface Analyzer” should I use x86 or x64?**

For x86 versions of Windows use x86 version of Attack Surface Analyzer. Likewise, for x64 versions of Windows use the x64 version of Attack Surface Analyzer.

**What internet browsers are supported?**

Attack Surface Analyzer is tested with latest versions of Internet Explorer, Chrome and Mozilla.

**How do I find out more information about the “Security Issues” reported in the Report.html file?**

On each of the section headers in the Report.html file there are “Explain…” links. Click on the “Explain…” links to open the corresponding section of the Help.html file that has more information. More information regarding the “Security Issues” can also be found in this document.

**How do I find out more information about the “Attack Surfaces” reported in the Report.html file?**

More detail regarding the Attack Surfaces can be found within this document and the Help.html file.

**Why was “Attack Surface Analyzer.exe” not installed on my system?**

When Attack Surface Analyzer is installed on a system without .NET Framework 4, Attack Surface Analyzer.exe is not installed. The command line options are provided through ASA.exe and will work, but the GUI program Attack Surface Analyzer.exe will not unless .NET Framework 4 is installed on the system before Attack Surface Analyzer is installed.

**What is the default report filename?**

The default report filename is created from the Product CAB filename. If you delete the Product CAB filename the report filename is cleared too. This is by design to ensure the report filename is unique by default.

**Why does the UAC prompt now show on some Windows Server installations?**

On Windows Server AAM-(Admin Approval Mode) is set at a lower setting for Server SKU's vs. Client SKU's. So there will be slightly different behavior in regard to UAC promptings for Server SKUs. You should expect the UAC prompts for Client SKU's even if the current account has admin privileges but for Server SKU's, only if the current account does not have admin privileges, then the UAC will ask for admin account authorization.

**Known Issues**

1. When scanning a system with >=100 GB of files, ASA will collect the data from the system. Attack Surface Analyzer will crash with "System.OutOfMemoryException" thrown. A workaround is to install Attack Surface Analyzer on a freshly built system to minimize the size of the data collected and analyzed.
2. In the Attack Surfaces tab of the Report.html file, if you collapse a section you cannot navigate to any of the subsection from the Table of Contents links. A workaround for this is to not collapse the sections in the report.html file.

**Test Results: Security Issues**

**Executables with Weak ACLs**

One or more executable files installed or changed by the application have weak access control lists (ACLs). Attack Surface Analyzer checks the ACL for each executable file that belongs to an administrator and that has been newly installed or has changed as a result of installing the application. The ACLs on these files must prevent non-administrators from modifying the files. Attack Surface Analyzer tests not only executable (.EXE) files, but also files that can contain executable content such as scripts and help files.

**Security Risks**

Executable files with weak ACLs are vulnerable to being modified by non-administrators. Possible modifications include changing the function of the executable content, assigning the file to a new owner, changing the attributes of the file, modifying the access rights of the file to give access to unauthorized accounts, and deleting the file altogether.

**Solution**

Weak ACLs are often the result of using default security settings in building the application. Change the settings to tighten the ACLs to eliminate permissions that allow non-administrators to tamper with the executable files. You can tighten the ACLs for the parent folder to secure all files within that folder. (See Directories Containing Objects with Weak ACLs.) If, however, the parent folder permissions cannot be tightened, you should set appropriate ACLs for each child folder or file.

In particular, do not allow any of the following access rights for non-administrator accounts:

* GENERIC\_ALL
* GENERIC\_WRITE
* WRITE\_OWNER
* WRITE\_DAC
* FILE\_WRITE\_ATTRIBUTES
* FILE\_WRITE\_EA
* FILE\_APPEND\_DATA
* FILE\_WRITE\_DATA
* DELETE

**Remarks**

Failure to protect executable code from write access by non-administrators is a common error. This exposes the code to the possibility of being compromised for the next user who executes the code on the machine. Weak ACLs allow non-administrators to alter an executable file. If an executable file has been altered, it might not work as expected. If the access rights are not correctly configured, an attacker could replace or alter the file's contents and cause it to behave maliciously.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[File Security and Access Rights](http://go.microsoft.com/fwlink/?LinkID=260658)

**Directories Containing Objects with Weak ACLs**

One or more objects contained within a directory have weak ACLs. Attack Surface Analyzer checks the ACLs for executables that have been created or changed as a result of installing the application and finds directories that contain executables with weak ACLs. Hierarchical ACLs (inherited ACLs) control access to all files and subdirectories within a directory. The ACLs on these directories must prevent non-administrators from modifying the subdirectories or the objects they contain.

Attack Surface Analyzer lists only the highest folder in a hierarchy that has the weak ACLs.

**Security Risks**

To protect an executable file, you must not only protect the file itself but also the directory in which it resides. Any account that has “folder: delete child”, “folder: add file” or “folder: add subfolder” permission on a directory has equivalent access as having write access to the executable file within that directory. These permissions allow a non-administrator to replace directories containing executable files with new directories containing new executable files or simply to delete directories and the executable files they contain.

**Solution**

When your application creates or changes executables, tighten the ACLs to eliminate permissions that allow tampering by non-administrators. Do not allow any of the following access rights for non-administrator accounts:

* GENERIC\_ALL
* GENERIC\_WRITE
* WRITE\_OWNER
* WRITE\_DAC
* FILE\_ADD\_FILE
* FILE\_ADD\_SUBDIRECTORY
* FILE\_WRITE\_ATTRIBUTES
* FILE\_WRITE\_EA
* FILE\_APPEND\_DATA
* FILE\_WRITE\_DATA
* FILE\_DELETE\_CHILD
* DELETE

In addition, you may need to change the **Inherited** flag in the root directory’s ACL to ensure that the tightened ACL applies to all subdirectories.

**Remarks**

Because Attack Surface Analyzer checks entire hierarchies for weak ACLs, the issue could be that a tighter ACL is not inherited by subdirectories and files. Therefore, check the inheritance settings in the ACLs for parent directories before changing the access rights in the subdirectories and files.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[EXPLICIT\_ACCESS Structure](http://go.microsoft.com/fwlink/?LinkId=259797)

[File Security and Access Rights](http://go.microsoft.com/fwlink/?LinkID=260658)

**Registry Keys with Weak ACLs**

One or more registry keys created or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs of registry keys that have been created or changed as a result of installing the application. The ACLs on these keys must prevent non-administrators from modifying the keys. Only administrators should have write access to the Local Machine registry hive.

**Security Risks**

Registry values can point to executable files, such as .EXE and .DLL files. Applications also use registry keys in the local machine registry hive to store or read the path of an executable file. If a non-administrator alters a key, such as by changing the path to lead to an untrusted executable, applications can execute the wrong file. For example, notepad.exe may normally load a file, good.dll. Even if this DLL is protected by appropriate ACLs, if notepad.exe can be directed to another file, bad.dll, and loads this file instead, the ACLs on good.dll become irrelevant. Modified registry keys can result in this sort of redirection.

**Solution**

When your application creates or changes registry keys, tighten the registry key ACLs to eliminate permissions that allow tampering by non-administrators. Do not allow any of the following access rights for non-administrator accounts:

* GENERIC\_ALL
* GENERIC\_WRITE
* WRITE\_OWNER
* WRITE\_DAC
* KEY\_SET\_VALUE
* KEY\_CREATE\_SUBKEY
* DELETE

**Remarks**

Attack Surface Analyzer checks the Registry hive for weak ACLs on .EXE and .DLL files. Avoid using the access rights listed above in order to tighten the registry key ACLs. This will restrict access to users that are non-administrators and help protect from a file lay-in-wait or elevation of privilege attacks.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Registry Key Security and Access Rights](http://go.microsoft.com/fwlink/?LinkId=260660)

**Processes with Weak ACLs**

One or more processes created or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs for processes that have been created or changed as a result of installing the application. The ACLs on these processes must prevent non-administrators from modifying the process.

**Security Risks**

Weak process ACLs could allow non-administrators to elevate their privileges and could give access to the system. Although this threat applies only to the local system – that is, it cannot be exploited remotely – this access could allow non-administrators to access sensitive user data or to change the behavior of the system.

**Solution**

When your application creates or changes processes, tighten the ACLs on the process to eliminate permissions that allow tampering by non-administrators. Do not allow any of the following access rights for non-administrator accounts:

* GENERIC\_ALL
* GENERIC\_WRITE
* WRITE\_OWNER
* WRITE\_DAC
* PROCESS\_CREATE\_PROCESS
* PROCESS\_CREATE\_THREAD
* PROCESS\_DUP\_HANDLE
* PROCESS\_SET\_INFORMATION
* PROCESS\_SET\_SESSIONID
* PROCESS\_VM\_OPERATION
* PROCESS\_VM\_WRITE

**Remarks**

Process ACLs have a set of permissions that allow users to control various aspects of the process. Although processes are not "named objects" in the normal Windows NT sense, they can be opened by ID through the **OpenProcess** Win32 API call.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Process Security and Access Rights](http://go.microsoft.com/fwlink/?LinkId=260661)

**Process Threads with Weak ACLs**

One or more processes threads created or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs for process threads that have been created or changed as a result of installing the application. The ACLs on these process threads must prevent non-administrators from modifying the thread.

**Security Risks**

Weak ACLs for process threads could allow non-administrators to elevate their privileges and could give access to the system. Although this threat applies only to the local system – that is, it cannot be exploited remotely – this access could allow non-administrators to access sensitive user data or to change the behavior of the system.

**Solution**

When your application creates or changes process threads, tighten the ACLs on the process thread ACLs to eliminate permissions that allow tampering by non-administrators. Do not allow any of the following access rights for non-administrator accounts:

* WRITE\_DAC
* WRITE\_OWNER
* THREAD\_SET\_INFORMATION
* THREAD\_SET\_CONTEXT
* THREAD\_IMPERSONATE
* THREAD\_SET\_THREAD\_TOKEN

**Remarks**

Process thread ACLs contain a set of permissions that allow users to control various aspects of the thread. Although process threads are not "named objects" in the normal Windows NT sense, they can be opened by ID through the **OpenThread** Win32 API call.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Thread Security and Access Rights](http://go.microsoft.com/fwlink/?LinkId=260662)

**Processes with NX Disabled**

One or more processes do not have NX enabled. Attack Surface Analyzer checks the memory layout of the process to verify that NX has been enabled. Applications must also opt-in to no-execute protection.

**Security Risks**

Marking memory regions as non-executable means that code cannot be run from that region of memory, which makes it harder for the exploitation of buffer overruns.

**Solution**

Enable the /NXCOMPAT option in the linker command when you build your application. This option is on by default in linker versions that support Data Execution Prevention (DEP). We recommend that you test your apps on a DEP-capable CPU and fix any failures you find that result from DEP.

**Remarks**

Data Execution Prevention (DEP) is a system-level memory protection feature that is built into the operating system starting with Windows XP and Windows Server 2003. DEP enables the system to mark one or more pages of memory as non-executable.

**More Information**

[Data Execution Prevention](http://go.microsoft.com/fwlink/?LinkId=260564)

**Services Vulnerable to Tampering**

One or more services created or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs for services in the Service configuration database that have been created or changed as a result of installing the application. The ACLs for these services must prevent not-administrators from modifying the service. This means that these services must not have weak ACLs on the binary path, the host DLL, the registry keys, or the service itself.

**Security Risks**

Services with weak ACLs are vulnerable to being redirected by a non-administrator to execute an untrusted file at startup. For example, a non-administrator could call **ChangeServiceConfig** to change the path of the service's executable file.

**Solution**

When your application creates or changes a service, tighten the ACL for the service to restrict non-administrators the access rights that allow them to change how the service runs. For information on tightening the ACLs for individual files, see Executables with Weak ACLs. Do not allow any of the following access rights to a service for non-administrator accounts:

* GENERIC\_ALL
* GENERIC\_WRITE
* WRITE\_OWNER
* WRITE\_DAC
* DELETE
* ACCESS\_SYSTEM\_SECURITY
* SERVICE\_CHANGE\_CONFIG

In addition, make sure that the directories for service-related files do not have weak ACLs. See Directories Containing Objects with Weak ACLs.

**Remarks**

Services have both generic and service-specific access rights associated with them. You must configure both categories of rights to restrict non-administrator accounts access that allows them to compromise the service. If the rights are not secured, an attacker could redirect the service to execute an untrusted file when it starts. For example, an attacker could call **ChangeServiceConfig** to change the path of the service's executable file.

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Modifying the DACL for a Service](http://go.microsoft.com/fwlink/?LinkId=260666)

**Services with Fast Restarts**

One or more services created or changed by the application allow fast restarts. Attack Surface Analyzer checks the **lpsaActions** elements of the **Service Failure Actions** structure for the **SC\_ACTION\_REBOOT** and **SC\_ACTION\_RESTART** values and finds services than can restart more than twice in 24 hours.

**Security Risks**

Services that are set to restart immediately after an abnormal termination can be used as an attack vector by repeatedly attempting to exploit the service until an attack is successful.

**Solution**

When your application creates or changes services, set that Reset Period for each service to prevent the service from restarting more than twice in a 24 hour period. If the product design requires the service have a fast restart, ensure the service has been thoroughly code reviewed and fuzzed.

**Remarks**

Fast restarts create a vulnerability related to Address Space Layout Randomization (ASLR). ASLR is a feature that loads executable code into random places in memory, making it more difficult to exploit security vulnerabilities. If non-administrators can make a service restart repeatedly, they can use brute force to load executable code into all possible locations and defeat ASLR.

**More Information**

["Address Space Layout Randomization (ASLR)" in Windows ISV Software Security Defenses](http://go.microsoft.com/fwlink/?LinkId=259799)

[ChangeServiceConfig2 function](http://go.microsoft.com/fwlink/?LinkId=260667)

**Vulnerable Named Pipes**

One or more named pipes created or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs for named pipes that have been created or changed as a result of installing the application. The ACLs for these named pipes must prevent non-administrators from modifying the pipe or creating new instances.

**Security Risks**

A common mistake with ACLs for named pipes is the unnecessary granting of **CreateInstance** to a non-administrator. This allows the non-administrator to create an instance of the pipe and put it in the queue used to service pipe clients, which could enable a number of exploitation scenarios.

**Solution**

When you application creates or changes named pipes, tighten the ACL for the pipe to restrict non-administrator accounts the access necessary to modify the pipe or create new instances. Do not allow any of the following access rights for non-administrator accounts:

* WRITE\_OWNER
* WRITE\_DAC
* FILE\_CREATE\_PIPE\_INSTANCE

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Named Pipe Security and Access Rights](http://go.microsoft.com/fwlink/?LinkId=260668)

**Vulnerable COM Classes**

One or more COM classes installed or changed by the application have weak ACLs. Attack Surface Analyzer checks the ACLs for COM classes that have been installed or changed as a result of installing the application.

**Security Risks**

A vulnerable COM control could grant an attacker access to the system at the user-level privilege.

**Solution**

When your application installs or changes a COM class, tighten the following ACLs to restrict unauthorized access (for the list or access rights to restrict, see Directories Containing Objects with Weak ACLs and Executables with Weak ACLs):

* Path to Inproc handlers
* Path to Inproc servers
* Paths to LocalServer and LocalServer32

In addition, tighten the ACL for the configuration registry key for both 32 and 64 bit views (for the list of access rights to restrict, see Registry Keys with Weak ACLs).

**Remarks**

The Microsoft Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact.

**More Information**

Access Control Lists

[Security in COM](http://go.microsoft.com/fwlink/?LinkId=260669)

**Vulnerable DCOM Classes**

One or more DCOM classes installed or changed by the application have weak ACLs. Surface Attack Analyzer checks the ACLs of DCOM classes installed or changed as a result of installing the application.

**Security Risks**

DCOM controls with weak ACLs have all the attack vectors of a COM control. (See Vulnerable COM classes.) In addition, these controls can be invoked remotely and require additional appropriate ACLs on their invocation.

**Solution**

When your application installs or changes DCOM classes, tighten the ACLs to restrict unauthorized access. Do not allow any of the following access rights for non-administrator accounts:

* Access (0x01)
* AccessLocal (0x02)
* AccessRemote (0x04)

In addition, tighten the ACLs on invocation to restrict unauthorized remote invocations of the classes. Do not allow any of the following invocation rights for non-administrator accounts:

* COM\_RIGHTS\_EXECUTE
* COM\_RIGHTS\_EXECUTE\_LOCAL
* COM\_RIGHTS\_EXECUTE\_REMOTE
* COM\_RIGHTS\_ACTIVATE\_LOCAL
* COM\_RIGHTS\_ACTIVATE\_REMOTE

**More Information**

[Access Control Lists](http://go.microsoft.com/fwlink/?LinkId=259796)

[Security in COM](http://go.microsoft.com/fwlink/?LinkId=260669)

**Memory Mapped Sections with Weak ACLs**

One or more memory-mapped sections created or changed by the application are marked as executable. Attack Surface Analyzer checks for memory-mapped sections that are unnecessarily marked as executable.

**Security Risks**

In the event of vulnerability that allows corruption of the address space of a process, such as a buffer overrun, technologies like Data Execution Protection (DEP), also known as NX, and the Visual C linker option /SafeSEH provide mitigations that can reduce the likelihood of code execution. These technologies make the best of a bad situation by terminating the process. This still results in a denial of service (DoS), but prevents code execution (EoP) that could lead to a worm or arbitrary payload execution. It is possible for an attacker to bypass these protections in some circumstances if the compromised process marks its memory pages as executable when they should not be. Marking a page for execution allows the attackers more freedom and flexibility to perform their attack. Any place marked as executable is a target for an attacker to place malicious code (if they have write access to the location) or is simply another location where the attacker can jump to.

**Solution**

When your application creates or changes memory-mapped sections, change the requested protection not to have PAGE\_EXECUTE. Pages are created by several APIs. As a result, to understand if you have correctly marked a page as execute or not, you will have to search your source code to find which calls you make that result in executable pages being created. To find these in your code base, search for the flag value (PAGE\_EXECUTE). There are few reasons to have execution access to a page not marked as MEM\_IMAGE, and as a result, it should be safe to remove the executable attribute from it.

**More Information**

[Managing Memory-Mapped Files](http://go.microsoft.com/fwlink/?LinkId=260670)

**Test Results: Attack Surface**

**System Information**

Running Processes

An application consists of one or more processes. A process, in the simplest terms, is an executing program. One or more threads run in the context of the process. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread.

[Processes and Threads](http://go.microsoft.com/fwlink/?LinkId=260563)

Processes and threads have a set of permissions that allow users to control various aspects about them. Though processes and threads are not "named objects" in the normal Win32 sense, they can be opened by id through the OpenProcess and OpenThread Win32 API calls.

Executable Memory Pages

Data Execution Prevention (DEP) is a system-level memory protection feature that is built into the operating system starting with Windows XP and Windows Server 2003. DEP enables the system to mark one or more pages of memory as non-executable. Marking memory regions as non-executable means that code cannot be run from that region of memory, which makes it harder for the exploitation of buffer overruns.

DEP prevents code from being run from data pages such as the default heap, stacks, and memory pools. If an application attempts to run code from a data page that is protected, a memory access violation exception occurs, and if the exception is not handled, the calling process is terminated.

DEP is not intended to be a comprehensive defense against all exploits; it is intended to be another tool that you can use to secure your application.

[Data Execution Prevention](http://go.microsoft.com/fwlink/?LinkId=260564)

Windows

In a graphical Windows-based application, a window is a rectangular area of the screen where the application displays output and receives input from the user. Therefore, one of the first tasks of a graphical Windows-based application is to create a window.

A window shares the screen with other windows, including those from other applications. Only one window at a time can receive input from the user. The user can use the mouse, keyboard, or other input device to interact with this window and the application that owns it.

[Windows](http://go.microsoft.com/fwlink/?LinkId=260566)

Kernel Objects

Kernel object handles are process specific. That is, a process must either create the object or open an existing object to obtain a kernel object handle. The per-process limit on kernel handles is 2^24. However, handles are stored in the paged pool, so the actual number of handles you can create is based on available memory. The number of handles that you can create on 32-bit Windows is significantly lower than 2^24.

Any process can create a new handle to an existing kernel object (even one created by another process), provided that the process knows the name of the object and has security access to the object. Kernel object handles include access rights that indicate the actions that can be granted or denied to a process. An application specifies access rights when it creates an object or obtains an existing object handle. Each type of kernel object supports its own set of access rights. For example, event handles can have set or wait access (or both), file handles can have read or write access (or both), and so on.

[Kernel Objects](http://go.microsoft.com/fwlink/?LinkId=260567)

Impersonation Tokens

Impersonation is the ability of a thread to execute in a security context that is different from the context of the process that owns the thread. When running in the client's security context, the server "is" the client, to some degree. The server thread uses an access token representing the client's credentials to obtain access to the objects to which the client has access.

The primary reason for impersonation is to cause access checks to be performed against the client's identity. Using the client's identity for access checks can cause access to be either restricted or expanded, depending on what the client has permission to do. For example, suppose a file server has files containing confidential information and that each of these files is protected by an ACL. To help prevent a client from obtaining unauthorized access to information in these files, the server can impersonate the client before accessing the files.

Access tokens are objects that describe the security context of a process or thread. They provide information that includes the identity of a user account and a subset of the privileges available to the user account. Every process has a primary access token that describes the security context of the user account associated with the process. By default, the system uses the primary token when a thread of the process interacts with a securable object. However, when a thread impersonates a client, the impersonating thread has both a primary access token and an impersonation token. The impersonation token represents the client's security context, and this access token is the one that is used for access checks during impersonation. When impersonation is over, the thread reverts to using only the primary access token.

[Impersonation](http://go.microsoft.com/fwlink/?LinkId=260680)

Windows Stations

A window station contains a clipboard, an atom table, and one or more desktop objects. Each window station object is a securable object. When a window station is created, it is associated with the calling process and assigned to the current session.

The interactive window station is the only window station that can display a user interface or receive user input. It is assigned to the logon session of the interactive user, and contains the keyboard, mouse, and display device. It is always named "WinSta0". All other window stations are non-interactive, which means they cannot display a user interface or receive user input.

When a user logs on to a computer using Remote Desktop Services, a session is started for the user. Each session is associated with its own interactive window station named "WinSta0".

[Window Stations](http://go.microsoft.com/fwlink/?LinkId=260569)

Desktops

A desktop has a logical display surface and contains user interface objects such as windows, menus, and hooks; it can be used to create and manage windows. Each desktop object is a securable object. When a desktop is created, it is associated with the current [window station](http://msdn.microsoft.com/en-us/library/windows/desktop/ms687096(v=vs.85).aspx) of the calling process and assigned to the calling thread.

Window messages can be sent only between processes that are on the same desktop. In addition, the hook procedure of a process running on a particular desktop can only receive messages intended for windows created in the same desktop.

The desktops associated with the interactive window station, Winsta0, can be made to display a user interface and receive user input, but only one of these desktops at a time is active

[Desktops](http://go.microsoft.com/fwlink/?LinkId=260570)

Modules

A module can be specified by its base address in the target's virtual address space, or by its index in the list of modules the engine maintains for the target. The module's index equals its position in the list of modules, and therefore this index will change if a module with a lower index is unloaded. All unloaded modules have indexes; these are always higher than the indexes of loaded modules. The base address of a module will not change as long as it remains loaded; in some cases it may change if the module is unloaded and then reloaded.

[Modules](http://go.microsoft.com/fwlink/?LinkId=260571)

**Service Information**

Services

The service control manager (SCM) maintains a database of installed services and driver services, and provides a unified and secure means of controlling them. The database includes information on how each service or driver service should be started. It also enables system administrators to customize security requirements for each service and thereby control access to the service.

A service program contains executable code for one or more services and can be configured to execute in the context of a user account from either the built-in (local), primary, or trusted domain. It can also be configured to run in a special service user account.

[About Services](http://go.microsoft.com/fwlink/?LinkId=260572)

When adding or changing system services you must ensure permissions on both the service object and associated service program restrict the ability for an unauthorized user or program to modify the service in order to elevate their privileges or have the system execute a malicious program on their behalf.

Drivers

A device driver is software that abstracts the functionality of a physical or virtual device. A device driver manages the operation of these devices. Examples of physical devices are network adapters, timers, and universal asynchronous receiver-transmitters (UARTs). An example of a virtual device is a file system. Implementing a device driver allows the functionality of your device to be exposed to applications and other parts of the operating system (OS).

The Windows security model is based on securable objects. Each component of the operating system must ensure the security of the objects for which it is responsible. Drivers, therefore, must safeguard the security of their devices and device objects.

[Windows Security Model: What Every Driver Writer Needs to Know](http://go.microsoft.com/fwlink/?LinkId=260573)

Since drivers run at system privilege, a compromised driver could grant the attacker a high level of privilege on the targeted system.

**ActiveX, DCOM, COM, File Registrations**

Registered ActiveX Controls

An ActiveX control is essentially a simple OLE object that supports the IUnknown interface. It usually supports many more interfaces in order to offer functionality, but all additional interfaces can be viewed as optional and, as such, a container should not rely on any additional interfaces being supported.

[Introduction to ActiveX Controls](http://go.microsoft.com/fwlink/?LinkId=260574)

ActiveX controls can be invoked by scripts in Internet Explorer and thus be attacked from remote scripts and have all the attack vectors of a COM control.

Registered DCOM Servers

Use the Distributed Component Object Model (DCOM) to allow your applications to be distributed across locations that make the most sense to you and to the application. The DCOM wire protocol transparently provides support for reliable, secure, and efficient communication between COM components such as ActiveX controls, scripts, and Java applets residing on different machines in a LAN, a WAN, or on the Internet.

[The Component Object Model](http://go.microsoft.com/fwlink/?LinkID=259801)

[Security in COM](http://go.microsoft.com/fwlink/?LinkId=260669)

DCOM controls can be invoked remotely and have all the attack vectors of a COM control.  They also have additional ACLs on the invocation.

DCOM Default Permissions

This section shows the default permissions set for DCOM.

Any changes to the default permissions should be reviewed to ensure there is no increase in attack surface.

Registered COM Controls

The Microsoft Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), as well as others. COM specifies an object model and programming requirements that enable COM objects (also called COM components, or sometimes simply objects) to interact with other objects. These objects can be within a single process, in other processes, and can even be on remote computers. They can be written in different languages, and they may be structurally quite dissimilar, which is why COM is referred to as a binary standard; a standard that applies after a program has been translated to binary machine code.

[The Component Object Model](http://go.microsoft.com/fwlink/?LinkID=259801)

[Security in COM](http://go.microsoft.com/fwlink/?LinkId=260669)

A vulnerable COM control could grant the attacker access to the system at the user level privilege.

File Registrations

Files with a shared common file name extension (.doc, .html, and so on) are of the same type. If you plan to associate one or more file types with a new application, then you must define a ProgID for each file type that you want to associate with the application.

[File Types](http://go.microsoft.com/fwlink/?LinkId=260575)

A new file extension usually indicates the presence of a file parser in a newly installed application. New file parsers should be fuzzed. Each file parser has an attack vector. Each file parser should be fuzzed to verify they are secured properly.

**Internet Explorer**

Attack Surface Analyzer examines Microsoft’s web browser, Internet Explorer, for possible security misconfigurations.  If you are using an alternative web browser as the default on your system please refer to your web browser vendor’s security best practice guidelines.

Pluggable Protocol Handlers

Asynchronous pluggable protocols provide the capability to create custom URL protocols.

[Asynchronous Pluggable Protocols](http://go.microsoft.com/fwlink/?LinkId=260577)

Silent Elevation Entries

The latest versions of Internet Explorer run in Protected Mode, which helps protect users from attack by running the Internet Explorer process with greatly restricted privileges.  Protected Mode significantly reduces the ability of an attack to write, alter or destroy data on the user's machine or to install malicious code.  Internet Explorer allows silent elevation of broker processes to medium integrity level by creating an elevation policy, which is a series of registry keys and values that tell Protected Mode how to handle elevation for a specific broker.

[Understanding and Working in Protected Mode in Internet Explorer](http://go.microsoft.com/fwlink/?LinkId=260578)

Any changes to the elevation policy should be evaluated to confirm they are necessary as they represent additional attack surface that may be vulnerable to exploitation.

Silent Elevation Entries (RunDLL32)

The latest versions of Internet Explorer run in Protected Mode, which helps protect users from attack by running the Internet Explorer process with greatly restricted privileges.  Protected Mode significantly reduces the ability of an attack to write, alter or destroy data on the user's machine or to install malicious code.  Internet Explorer allows silent elevation of broker processes to medium integrity level by creating an elevation policy, which is a series of registry keys and values that tell Protected Mode how to handle elevation for a specific broker.

[Understanding and Working in Protected Mode in Internet Explorer](http://go.microsoft.com/fwlink/?LinkId=260578)

Any changes to the elevation policy should be evaluated to confirm they are necessary as they represent additional attack surface that may be vulnerable to exploitation.

Preapproved Controls

Adding a control to the pre-approved list will attract the attention and scrutiny of the security research community.  Any vulnerability found in these controls would expose a significantly larger number of users than if the control were not on the pre-approved list.  Keeping controls off the pre-approved keeps users secure by default and protects customers who do not require this control.

[ActiveX Security: Improvements and Best Practices](http://go.microsoft.com/fwlink/?LinkId=260579)

Browser Helper Objects

A browser helper object (BHO) is a lightweight DLL extension that adds custom functionality to Internet Explorer.  BHOs typically do not provide any user interface (UI) of their own.  Rather, they function in the background by responding to browser events and user input.  For example, BHOs can block pop-ups, auto-fill forms, or add support for mouse gestures.  It is a common misconception that BHOs are required by toolbar extensions.

BHOs are convenient tools for end users and developers alike; however, because BHOs are granted considerable power over the browser and Web content, and because they often go undetected, users should take great care to obtain and install BHOs from reliable sources.

[Building Browser Helper Objects with Visual Studio 2005](http://go.microsoft.com/fwlink/?LinkId=260580)

IE Zones

Adding sites to Internet Explorer zones may increase the attack surface of the browser.  For example, if you add a site to the Trusted Sites zone and an attacker is able to exploit a cross site scripting vulnerability on that site, then the attacker will be able to run script in the Trusted Sites context.  Depending upon how the user has configured their browser security settings this may represent a vulnerability.

[Default URL Security Zones](http://go.microsoft.com/fwlink/?LinkId=260581)

IE Actions Policy

Each URL security zone has a set of URL actions, with a URL policy assigned to each action. The URL actions cover all operations that have security implications. The URL policy assigned to each URL action determines how that URL action is handled.

[URL Actions and Policies](http://go.microsoft.com/fwlink/?LinkId=260601)

**Network Information**

Ports

When a client initiates a TCP/IP socket connection to a server, the client typically connects to a specific port on the server and requests that the server respond to the client over an ephemeral, or short lived, TCP or UDP port.

[TCP/UDP](http://go.microsoft.com/fwlink/?LinkId=260604)

An opened port generally indicates an application running on a target computer that accepts input from a remote source.

Named Pipes

A named pipe is a named, one-way or duplex pipe for communication between the pipe server and one or more pipe clients. All instances of a named pipe share the same pipe name, but each instance has its own buffers and handles, and provides a separate conduit for client/server communication. The use of instances enables multiple pipe clients to use the same named pipe simultaneously.

[Named Pipes](http://go.microsoft.com/fwlink/?LinkId=260625)

A named pipe with a weak ACL could allow an attacker to compromise an application using the named pipe as an input.

RPC Endpoints

Microsoft Remote Procedure Call (RPC) defines a powerful technology for creating distributed client/server programs. The RPC run-time stubs and libraries manage most of the processes relating to network protocols and communication. This enables you to focus on the details of the application rather than the details of the network.

[Remote Procedure Call](http://go.microsoft.com/fwlink/?LinkId=260626)

An RPC server can be compromised remotely, giving an attacker remote access to the targeted system.

Network Shares

A shared resource is a local resource on a server (for example, a disk directory, print device, or named pipe) that can be accessed by users and applications on the network.

[Configure a Network Share Item](http://go.microsoft.com/fwlink/?LinkId=260689)

Network Interfaces

The section lists the addition or removal of networking adaptors on the system.  This information is used in conjunction with installed protocols and the analysis of Windows Firewall settings to identify changes to remotely reachable services.

[High Performance Network Adapters and Drivers](http://go.microsoft.com/fwlink/?LinkId=260629)

**Firewall**

A firewall provides an additional layer of protection against network based attacks.  The absence of a firewall increases the attack surface of computers that can communicate via a network.  Attack Surface Analyzer is only able to confirm the settings and security implications of the inbuilt Windows Firewall.  These checks are irrelevant if a third party Firewall product is installed.

Firewall Rules

The firewall helps protect the device on which it runs by blocking network traffic at the IP and transport layers of the OSI model.  Firewall rules are used to specify what traffic is permitted and what traffic will be denied.

[Firewall Rules and Requirements](http://go.microsoft.com/fwlink/?LinkId=260631)

Firewall Profiles

A firewall profile is a way of grouping settings, such as firewall rules and connection security rules that are applied to the computer depending on where the computer is connected.  On computers running newer version of Windows there are three profiles for Windows Firewall with Advanced Security.  Only one profile is applied at a time.

[Understanding Firewall Profiles](http://go.microsoft.com/fwlink/?LinkId=260632)

Firewall Service Restriction Rules

This lists changes to services which have firewall restriction rules in place.  Restriction rules specify services to be restricted when sending or receiving network traffic.  The Windows Service Hardening rules collection can contain rules which can allow this service specific inbound or outbound network access per specific requirements.

[Restricting Service (An example program restricting access using the Firewall APIs)](http://go.microsoft.com/fwlink/?LinkId=260633)

Authorized Applications

Application rules allow dynamic edge traversal using the Windows Firewall with Advanced Security APIs.

[Adding an Application Rule to Allow Dynamic Edge Traversal](http://go.microsoft.com/fwlink/?LinkId=260634)

**System Environment, Users, Groups**

%PATH% Entries

A lay-in-wait issue is present if an attacker can influence files in someone else’s %PATH%. Inspection of the system %PATH% reveals to the attacker how the users and the system search for dlls and other binaries during loading. If the attacker can modify a binary in the path, or add a binary to a folder in the path, the prospects of getting a rogue executable loaded are very good.

[User Environment Variables](http://go.microsoft.com/fwlink/?LinkId=260635)

Accounts

Accounts can be created for users, services and other operations on the computer.  Additions or changes to accounts are listed here and should be reviewed against your threat model.  Accounts can have security privileges or permissions granted to them through access control lists (ACLs).

[About User Profiles](http://go.microsoft.com/fwlink/?LinkId=260674)

Groups

Groups can have security privileges or permissions granted to them through access control lists (ACLs).  Ensure changes to groups are essential and assess the impact on your attack surface.

[User groups in Windows (User Documentation)](http://go.microsoft.com/fwlink/?LinkId=260675)

Group Membership

This table lists the changes in group membership on the computer.

[User groups in Windows (User Documentation)](http://go.microsoft.com/fwlink/?LinkId=260675)

Account Privileges

User right/privilege assignments can be found in the Local and Domain Security Policy GUI. This table contains a list of total accounts on the system, and new accounts created with the installation of the application.

[Default Privileges](http://go.microsoft.com/fwlink/?LinkId=260676)

**Additional Information**

**Contact Us**

<http://social.msdn.microsoft.com/Forums/en-US/sdlprocess/threads>

**Other Resources**

* Microsoft SDL Blog: <http://www.microsoft.com/security/sdl/default.aspx>
* Microsoft Download Center Link: <http://www.microsoft.com/en-us/download/details.aspx?id=24487>

© 2012 Microsoft.  All rights reserved.

This document is made available to you without warranty of any kind and confers no rights on you whatsoever except the right to review it solely as a reference. The entire risk of use or results from use of this document remains with you.