UNIVERSIDAD POLITECNICA DE VALENCIA

Local Protection in Windows Server Systems

Apellidos, nombre	Terrasa Barrena, Andrés (aterrasa@dsic.upv.es), Espinosa Minguet, Agustín (aespinos@dsic.upv.es)
Departamento	Dpto. de Sistemas Informáticos y Computación
Centro	E. T. S. de Ingeniería Informática

UNIVERSIDAD POLITECNICA DE VALENCIA

1. Key Concepts

This document introduces the fundamentals of **local protection** in Windows Server systems, which is based on a few basic abstractions and mechanisms:

- The system incorporates two abstractions in order to identify and authorize the people (or *users*) that can log in to use the system: user and group accounts.
- The system stores some protection attributes in any running process, and it also stores some other attributes in the system itself and its resources (folders, files, printers, etc.).
- The system enforces some protection rules which control the actions each process is allowed to do in the system, which may affect particular resources and/or the entire system.

The system administrator is the person in charge of configuring the system in such a way that the right people can use the system in the right way. In order to do so, the administrator needs not only to understand the abstractions and mechanisms above, but also to effectively use the specific system tools available for this configuration. In the case of Windows Server, these tools are normally graphical applications named *consoles*. However the use of these tools are outside the scope of this document, which is focused on the protection concepts rather than its related procedures or tools.

2. Objectives

After reading this article, students will be able to:

- Identify the main features of user and group accounts in Windows Server systems, including the attributes which may be edited and the ones which the system configures automatically.
- Identify the main built-in user and group accounts in the system, and understand their main characteristics.
- List the protection attributes in the Windows Server protection model, including the attributes of processes (SAT), the system (user rights) and the system's resources (permissions); understand how these protection attributes are compared in order to enforce the system protection.
- Understand the protection rules in Windows Server systems, including the ones that control the initial association of permissions to resources and the conditions under which this association can be later modified.
- List the standard and individual permissions in Windows Server, and understand their relationship.

3. Introduction

Local protection is a topic in System Administration, which is a knowledge area inside Computer Science. The aim of local protection is to configure a computer (or more specifically, the computer's operating system) in order to ensure that the computer is used by the right people and in the right way; or, more specifically, that: (1) only certain people (or users) can access the computer, and (2) when any user is working with the computer, this user cannot compromise, either accidentally or intentionally, the integrity of any other user's work or the system itself.

In this context, the term “local” is used to denote that, despite the fact that the computer may be connected to other computers, the rules by which protection is enforced are local to that computer (i.e., they do not involve others). This is the most basic case of protection, and it is key to understand other types of protection rules that may be applied to many computers at once (which are outside the scope of this document).

In particular, this document introduces the local protection rules of Windows Server systems. Windows Server is a close-source operating system by Microsoft which integrates a set of abstractions and tools for system administration. Several of the concepts here introduced are common in the different versions of Windows Server, including Windows 2000, Windows Server 2003 and Windows Server 2008, unless explicitly indicated otherwise.

4. User Account

Windows Server 2003 calls *user* each person that can enter, or *log on* to, the system to work with it. In order to control the logon process and the actions of each user, Windows Server incorporates the *user account* abstraction. A user account is a repository where the system stores all the information about each user. The most important items inside the user account are the following:

- **Username**. This is the name by which the user is *identified* in the system. Each user must have a different username for the identification to be univocal.
- **Full name**. Is the user's full name.
- **Password**. This is a coded word which is the default mechanism for *authenticating* the user name during the log in process, that is, only users who are identified and can be positively authenticated are *allowed* to log in to the system. In Windows Server, the password is case sensitive.
- **Home directory**. This is the directory where all the user's personal files will reside. The home directory for each user is private: no one else can enter the directory, unless the the user (owner) grants other users some permissions.
- **Disable**. This feature can be used to temporarily disable an account. A disabled account still exists, but cannot be used for accessing the system, not even knowing the password.

There is a special item associated with each account, but unlike the previous ones, it cannot be manually configured. This item is called the **Secure Identifier** or SID . This identifier is internal to the system, and it is automatically generated when the user account is created. Furthermore, SIDs are generated in such way that they are unique, meaning that it is ensured by definition that there cannot be two equal SIDs in the same (or in any other) Windows Server system. Windows Server always uses the SID (and not the username) in any internal action related to protection (e.g., whether or not a user has sufficient permissions to run a program, or to access a resource). The advantage of this model is that the SID is a piece of data completely private to the operating system, in the sense that no user (not even the system administrator) can set it anywhere in the system. This enhances system security, since no user can get a higher degree of privileges in the system by means of *supplanting* the identity of another user.

When a Windows Server 2003 system is installed, two user accounts are created by default (these are normally called *built-in accounts*): "Administrator" and "Guest". The **Administrator** is a special user, the only one with administrative powers by default. Such administrative powers include the full management of the system in all the aspects in which it is configurable: users, groups, passwords, resources, etc. The Administrator account cannot be deleted or disabled. On the other hand, the Guest account is normally used by those people who do not have their own user accounts to log in to the system. Typically, this account has no password assigned, since it is assumed that the privilege level associated with it is minimum. In any case, the administrator is advised to disable this account if it is not needed in the system.

5. Group Account

The security information stored on a user account is sufficient to establish the degrees of freedom (or otherwise, the restrictions) that each user must have in the system. However, it may be sometimes tedious for the administrator to determine such restrictions in a per-user basis, especially in systems with a large number of users. The concept of *group* allows the system administrator to logically group any number of users in the system, and then to set permissions and restrictions to the entire group at once. A user can belong to as many groups as necessary, implicitly having the *sum* of the permissions granted to all of such groups. The protection strategy is much more flexible and powerful when it is based on groups rather than on individual users.

Consider, for example, that a company's system is used by employees of different ranks, and each rank has a different privilege level. Suppose you want to change the rank of an employee due to a promotion, for example. If protection was based on individual users, changing the privilege level of this user would imply revising the permissions on all the resources in the system, and to change the permission of the user according to the user's new rank (which can be a tedious and error-prone process). On the contrary, if the protection strategy was based on groups, this operation would be as simple as changing the user from one group to another. Therefore, Windows Server recommends that permissions are assigned to *groups*, and not to individual users, whenever possible.

As it happens for user accounts, there are a number of *built-in groups* in any Windows Server 2003 system:

- **Administrators**. It includes all users who must possess full administrative rights in the system. Initially, this group has a single user in it (the "Administrator").
- **Guests**. It includes the guest users of the system. Initially, it has has the "Guest" account as its sole member.
- **Users**. It includes the regular users of the system. They have the right to access the system both interactively (log on) and through the network.
- **Backup Operators**. These users can make (and restore) a copy of the entire system.
- **Power Users**. These are users with certain administrative rights. They can change the system time, create user and group accounts, share folders and printers, etc.

The Administrator can make each user to belong to any number of such built-in groups, as well as crate new groups in order to refine this initial structure, according to the needs of the organization where the system is located.

Finally, Windows Server also defines a series of the so-called *special identities*, which are similar to, but not exactly the same as, groups. A special identity corresponds to a set of users that match some predefined rule. Like groups, special identities can be granted permissions in the system, but unlike groups, their members cannot be set up manually (the membership is automatically determined by the system, depending on the user matching or not the rule by which the identity is defined). The main special identities defined in a Windows Server system are the following:

- **Everyone**. This identity groups all users which are known by the system. It always includes users from the local computer and it may also include users existing in other computers (if the computer is included in a *domain*). From Windows Server 2003 on, this group does *not* include anonymous connections (connections without a username and a password).
- **Authenticated Users**. This identity represents all users which have their own account to log on to the system. Thus, users which may be logged on by using the "Guest" account do not belong to this group, although they do belong to "Everyone".
- **Interactive**. This identity represents all users who have the right log on locally to the system.
- **Network**. This identity represents all users who have the right to access the computer from the network.

6. Protection Model

The security model of Windows Server defines how the system carries out the *access control* on each user and each group. In other words, it is the model followed by the system in order to establish the actions that a user (or group) is authorized to perform. This model is based on a matching process between certain *protection attributes* that are assigned to user processes on the one hand, and the system and its resources on the other. In the case of the system and its resources, Windows Server defines two different and complementary concepts: *user right* and *permission*, respectively.

The following sections detail the protection attributes of user processes (Section 7), the user rights which can be set up in the system (Section 8) and the protection attributes of the system resources (Section 9). Finally, Section 10 provides the specific rules by which the system carries out the access control of processes.

7. Protection Attributes of Processes

When a user is authorized to log on to a Windows Server system, the system builds a security pass called *Security Access Token* or **SAT**. This token contains information about the user protection, and Windows Server includes it in the process that is created during the user's logon process (normally, this first process runs the user's "Desktop"). Then, the SAT is also automatically copied to any other process that may be created by this first one afterwards. Thus, the user's *protection attributes* are present in every process run by the user, and they are used to control the actions that any of these processes makes to the system and its resources on behalf of the user.

Specifically, the SAT contains the following protection attributes:

1. **SID**. The unique internal identifier of the user.
2. **SIDs of the user's groups**. List of the SIDs of the groups that the user belongs to.
3. **User rights**. List of the user's granted user rights. This list is built by including all the rights that the user has granted, either individually or through the membership of some group (see Section 8 below).

This way of building the security token introduces one of the mantras of Windows Server's protection: the access level of a user implicitly includes the levels of all the groups which the user belongs to.

8. User Rights

A *user right* is a protection attribute of the system conferring a particular set of users/ groups the ability to perform an action which affects the entire system (as an opposite to a particular resource, which protection is controlled by *permissions*). As explained above, the list of granted user rights is explicitly added to the security pass (SAT) that the system builds when the user logs on to system. This list includes the rights granted to both the user and any of the user's groups.

There is a predefined, fixed list of user rights (the Administrator cannot modify this list), which Windows Server separates into two categories: *logon rights* and *privileges*. The first ones set up the various ways in which a user can connect to the system (mainly, interactively or through the network). while the latter ones refer to certain predefined actions that the user can perform while being connected to the system. Table 1 summarizes the main examples of each type, along with their descriptions.

LOGON RIGHTS	
Name	**Meaning**
Access this computer from the network	Allows/prevents users from connecting to the computer from another computer over the network.
Logon locally	Allows/prevents the user to interactively log on to computer.
PRIVILEGES	
Name	**Meaning**
Add workstations to the domain	Allows users to add computers to the current domain.
Backup/restore files	Allows users to make/restore backups of files and folders.
Traverse Folders	Allows users to access files on which they have permissions through a path of folders over which they may have no permissions.
Change the system time	Allows users to modify the computer's internal clock.
Install device drivers	Allows users to install and uninstall Plug and Play device drivers.
Shut down the system	Allows users to turn off the local computer.
Take ownership of files and other objects	Allows users to take ownership (become the new owner) of any object with security attributes (including files, folders, Active Directory objects, etc .).

Table 1. Main user rights in Windows 2003

It is important to note that, when there is a *conflict* between an action being granted by a permission and denied by a user right, or vice-versa, the user right always prevails. For example, any member of the Backup Operators group has the right to back up all files in the system, including the ones in which the user does not have any permission. In this case, the user would not be able to access any of these files directly (by using the Windows Explorer, for example) but she can copy them while performing a backup. Similarly, the Administrator has the right to take ownership of any file, including those files on which this user has no permissions.

9. Protection Attributes of Resources

In Windows Server, each object in a NTFS file system (that is, each folder or file) has the following protection attributes:

1. **Owner's SID**. Each resource in Windows Server has an owner, which is internally represented by the owner's SID. Initially, the owner of a folder/file is always the user who created it, but this attribute can be later changed under certain conditions (as explained below).
2. **Protection Access Control List**. This list includes the *permissions* that users have on the file or folder. The list may contain an indeterminate number of entries, with each one granting or denying a particular set of permissions to a given user or group. As a result, Windows Server allows each object in the file system to incorporate multiple access levels, and each level may be *positive* (a permission is granted) or *negative* (a permission is denied).
3. **Security Access Control List**. This second list is used to define what actions on a file or folder must be *audited* by the system. A brief description of this is given here, since this issue is outside the scope of this document. The audit process writes entries in the system's *security log* about the actions that users perform on files or folders (these entries can later be accessed by using an administrative tool called Event Viewer). The system only audits the actions specified in the security list of each folder or file (and only from the specified users and/or groups). This list is initially empty for all files and folders in the file system (meaning that no object in the filesystem is audited by default).

The protection access control list is actually divided into two lists, each named *Discretionary Access Control List* or **DACL**. Each element in a DACL is called Access Control Entry or ACE. Each entry links a particular user's or group's SID to a positive (granted) or negative (denied) permission, as explained above. The actual permissions that may be granted or denied to a SID in Windows Server are later described in Section 9.2.

The fact that every object has two DACLs instead of one is related to the *permission inheritance* mechanism of Windows Server: each object can implicitly inherit the permissions of its parent folder. In addition, it can also define its own permissions (called *explicit* permissions in Windows Server). In other words, each object can potentially have an *inherited* DACL and an *explicit* DACL (although any of them, or both, can be void, as explained below). In this way, the explicit permissions defined for any given folder, plus any inherited permissions that this folder may have, become the list of inherited permissions of every one of the folder's files and subfolders, and so forth. This inheritance mechanism is dynamic, in the sense that any change in the definition of an explicit permission of a folder (the permission is added, edited or removed) is automatically reflected on the corresponding inherited permissions in the folder's subfolders and files.

9.1. Associating Permissions to Resources

The association of permissions to files and folders follows a series of rules:

1. When you create a **new** file or folder, it does not have any explicit permission and it receives the entire list of permissions (both explicit and inherited) of its parent folder as inherited permissions.
2. If you want to **add** permissions to a file or folder, such permissions can be added to the explicit list only. In the same way, you can only edit or delete an *individual* permission if this permission is explicit.
3. The control over the **inheritance** of permissions (i.e., which resources do inherit and which permissions are inherited) can be done at the two following levels, independently:
   - Each *object* in the filesystem can be configured to inherit or not the permissions of its parent folder. As explained above, this type of inheritance is enabled by default when the object is created, but it can be disabled if necessary. If so, the inherited DACL is disabled, meaning that all the permissions defined in the object's parent folder cease to be applied to the object. However, it is important to note that the inheritance configuration of an object is dynamic: it can be enabled or disabled at will, any number of times, without affecting the explicit permissions that may be defined for the object.
   - Each *permission* contains a rule that specifies which objects below the current object in the filesystem hierarchy will inherit the permission (that is, on which of these "descendant" objects the permission will be applied). This rule only takes effect if the object on which the permission is being configured is a folder, since only folders can contain "descendant" objects. Therefore, when configuring an *explicit* permission on any given folder F, you can restrict on which objects below F the permission will be applied. Specifically, the available options are the following: (a) on the F (current) folder only, (b) on F's children folders (subfolders) only, (c) on F's children files only, or any combination of these three options. The default rule when creating an explicit permission is to let the permission be inheritable by the folder, subfolders and files.
4. When you **copy** a file or folder to another location in the filesystem, this is considered a creation, and therefore the new (copied) file/folder gets an empty list of explicit permissions and the inheritance enabled. As a result, the copied file/folder will get its inherited permissions from its new parent folder.
5. When you **move** a file or folder to another location in the filesystem, Windows Server distinguishes two cases: if you move the object within the *same* same NTFS volume (partition), the inheritance is disabled and the object *retains* all its original permissions as explicit permissions in the new location. However, if the destination volume is different, then the system follows the rule for copy explained above: the original permissions are lost, and the object only gets the permissions from its new parent folder as inherited.

9.2. Standard and Individual Permissions

Windows Server incorporates a list of *standard permissions* that can be granted (or denied) to folders and files. Several of these permissions are equally named for folders and files, but with different semantics in each case, as explained below. Standard permissions are sometimes called "permission templates", since they are actually pre-defined combinations of some other fine-grained permissions, called *individual permissions*, which control each individual action that can be performed on folders and files. Thus, standard permissions are logical combinations of individual permissions with a clear and useful semantics, which are provided to make the permission management easier for the Administrator (or any other user which wants to manage her own files). Only in rare occasions it is necessary to define a "special" permission (one that it is not standard), by configuring each of the 13 possible individual permissions.

Table 2 shows the list of standard permissions available for folders and files, along with the semantics of each one. For the sake of simplicity, the description in the table refers to the permissions being granted (positive permissions), but please remember that permissions can also be negative in Windows Server, in which case the semantics would be the opposite. As you can see in the table, some of the standard permissions are defined *incrementally*, so that some permissions include some other(s) in their definitions. It is also important to note that the inheritance of permissions between a parent folder and its children subfolders and files is defined naturally: subfolders and files inherit the same permissions set in their parent folder, except in the case of the `List` permission, which is inherited by (sub)folders, but not by files.

FOLDERS	
Name	**Description**
`List`	Allows listing the folder: show the files and subfolders that it contains.
`Read`	Allows listing the folder and reading the contents of its children files and subfolders along with their owners, permissions and attributes (system, read only, hidden, etc.).
`Write`	Allows the creation of new files and subfolders in the folder. Allows modifying the attributes of the folder, as well as viewing the owner, permissions and attributes.
`Read and Execute`	Allows the access to the hierarchy of subfolders below the folder, even if you do not have permissions on them. It also includes all the permissions of `Read` and `List`.
`Modify`	Allows the deletion of the folder plus all the permissions inside `Write` and `Read and Execute`.
`Full Control`	Allows changing permissions and taking ownership of the folder, deleting any of the folder's void subfolders and files (even without permissions on them), plus all the previous permissions.
FILES	
Name	**Description**
`Read`	Allows reading the contents of the file, along with its ownership, permissions and attributes (system, read-only hidden, etc.).
`Write`	Allows change/overwrite the contents of the file, modify its attributes and read its owner, permissions and attributes.

FILES	
Name	**Description**
`Read and Execute`	Allows the execution of the file (if it is executable), and the permissions included in `Read`.
`Modify`	Allows deleting the file, plus all the permissions included in `Write` and `Read and Execute`.
`Full Control`	Allows changing permissions and taking ownership of the file, plus all the previous permissions.

Table 2. Standard permissions of folders and files in Windows Server

In cases when none of the standard permissions matches the access level required in a folder or file, a "special" or "advanced" permission needs to be configured instead. In this case, the permission is configured by granting/denying each of the 13 possible individual permissions independently, which are described in Table 3.

Name	Description
`Traverse Folder/Execute File`	Applied to a folder, it allows using its subfolders (in which you may have no permissions) in order to access an object (in which you do have permissions). Applied to a file, it allows executing it.
`List Folder/Read Data`	Applied to a folder, it allows showing the folder's contents (subfolders and files). Applied to a file, it allows reading its contents.
`Read Attributes`	Allows reading the regular NTFS attributes of the folder/file (e.g., `hidden`).
`Read Extended Attributes`	Allows reading the extended (application-defined) NTFS attributes of the folder/file.
`Create Files/Write Data`	Applied to a folder, it allows the creation of files inside. Applied to a file, it allows editing and overwriting its contents.
`Create Folders/Append Data`	Applied to a folder, it allows the creation of subfolders inside. Applied to a file, it allows the addition of data at the end of it.
`Write Attributes`	Allows modifying the regular NTFS attributes of the folder/file.
`Write Extended Attributes`	Allows modifying the extended (application-defined) NTFS attributes of the folder/file.
`Delete`	Allows deleting the folder or file.
`Delete Subfolders and Files`	It only applies to folders. It allows for the deletion of the folder's (void) subfolders and files, even without having the `Delete` permission on them.
`Read Permissions`	Allows reading the permissions of the folder or file.
`Change Permissions`	Allows modifying the permissions of the folder or file.
`Take Ownership`	Allows taking ownership (become the owner) of the folder or file.

Table 3. Individual permissions in Windows Server

Finally, Table 4 shows the association between individual and standard permissions (in the table, standard permissions are set in columns, and referred to as by using their initials). As a curiosity, you can see that the list of individual permissions for `List` and `Read and Execute` are the same. In fact, what distinguishes them is how they are inherited: the first is inherited by folders only, while the second is inherited by folders and files.

Permission	FC	M	R&E	L	R	W
Traverse Folder / Execute File	+	+	+	+		
Read Folder / Read Data	+	+	+	+	+	
Read Attributes	+	+	+	+	+	
Read Extended Attributes	+	+	+	+	+	
Create Files / Write Data	+	+				+
Create Folders / Append Data	+	+				+
Write Attributes	+	+				+
Write Extended Attributes	+	+				+
Delete	+	+				
Delete Subfolders and Files	+					
Read Permissions	+	+	+	+	+	+
Change Permissions	+					
Take Ownership	+					

Table 4. Matching between standard and individual permissions in Windows Server

9.3. Modification of Protection Attributes

The rules that Windows Server enforces in order to control which users and groups can modify the protection attributes of a resource are fully integrated within the security model, which is based on the *permissions* and *rights* that the user/group may have on the resources in question. This is different from other operating system (e.g., Unix), which incorporate separate rules for the modification of the protection attributes.

Specifically, the rules that control which users/groups can change the protection attributes of resources (files and folders) are:

1. **Owner**. Any user/group granted with either the `Take ownership` individual permission (included in `Full Control`) on the resource, or the `Take ownership of files or other objects` user right, may become the new owner of the resource. In addition, from Windows Server 2003 on, the ownership can also be *given* to other users, but only by users having the user right `Restore files and Folders` (which is granted by default to built-in groups Administrators and Backup Operators).
2. **Protection Access Control List**. Any user/group granted with the `Change Permissions` individual permission (included in `Full Control`) on the resource can modify the permissions of the resource. Independently, the *owner* of a re-

source can always change its permissions. The specific actions included in the permission change of a resource are: (a) enable/disable the inheritance of permissions from the parent folder, and (b) modification (add, remove, edit) of explicit permissions on the resource

3. **Security Access Control List**. The rules are the same as in the previous case.

At this level, it is interesting to analyze the differences of the Windows Server and Unix security models, regarding the Administrator figure (called "root" in Unix systems). In the Unix world, the user "root" can perform by definition any action in the system, without restrictions. However, Windows Server enforces the same protection rules on every user, including the Administrator. For example, if the Administrator has no permissions on a resource, she will not be able to access it. The Administrator would take ownership of the resource (thanks to the *user right* mentioned above) and then grant permissions to herself. So, the Administrator has more granted permissions and user rights than other users (in order to effectively manage the system), but still is controlled by the same protection rules as any other user.

10. Protection Rules

Windows Server applies the following rules when a user process wants to perform some action on a folder or file:

- An **action** of a process (e.g., double click on a text file icon) may involve several individual actions on multiple files and/or folders (following the example, the action would imply two sub-actions: execute the program associated with the file, and opening the file, at least for reading). If so, the system checks whether or not the process has enough permissions for executing all of these individual actions. If any single permission is not granted to the process, the entire action is rejected and a generic "deny" error is issued.

- Permissions are **cumulative**: a user process user implicitly incorporates all the permissions granted to any SID included in its SAT (see Section 7), i.e., the permissions granted to both the user owner of the process and all the user's groups.

- If a process **lacks** some permission on a file/folder (none of its SIDs has the permission granted), then the process will not be able to perform the corresponding action on the file/folder.

- In case of a **conflict** between two permissions which apply to the same process on a given folder/file (i.e., one of the process' SIDs has some permission granted while some other SID has the same permission denied), negative permissions take precedence over positive permissions, and explicit permissions take precedence over inherited permissions.

These rules are easier to understand if you take into account the particular algorithm that Windows Server uses to grant or deny a concrete action on a given folder or file. The algorithm traverses sequentially the protection DACLs of the folder/file, until one of the following conditions are met:

1. Each permission involved in the requested action is explicitly **granted** to any of the process' SIDs. In this case, the action is allowed.
2. Some permission involved in the requested action is explicitly **denied** to any of process' SIDs. In this case, the action is rejected.
3. The traversal of the DACL)s **ends without finding** an entry which relates any of the process' SIDs with the requested action. In this case, the action is rejected.

This algorithm actually produces the behavior described by the above rules because the order in which Windows Server orders the entries in the DACLs of each object. This order is always as follows: negative explicit permissions, positive explicit permissions, negative inherited permissions and positive inherited permissions.

11. Concluding Remarks

This document has presented the fundamentals of local protection in Windows Server systems. After reading it, you should be capable of understand the basics about user and group accounts, and how to effectively configure the user rights and the permissions in the system, in order to create a safe and productive environment on which several users can work either in common or on their own.

In combination with the abstractions mentioned above, Windows Server provides the system administrator with a set of system tools, normally in the form of graphical consoles, to configure the protection of the system. Although the concrete appearance of such tools is outside the scope of this document, they are designed to match the explained abstractions, and so it should be reasonably easy to learn how to use them after reading this document.

12. Bibliography

[1] Spealman, J; Hudson, K; Craft, M.: MCSE Self-Paced Training Kit (Exam 70-294): Planning, Implementing, and Maintaining a Microsoft(r) Windows Server(tm) 2003 Active Directory(r) Infrastructure. Microsoft Press, 2004.

[2] Holme, D.; Thomas, O.: MCSA/MCSE Self-Paced Training Kit (Exam 70-290): Managing and Maintaining a Microsoft Windows Server 2003 Environment: Managing and Maintaining a Microsoft Windows Server(tm) 2003 Environment. Microsoft Press, 2004.