Global Information Assurance Certification Paper

Protecting Administrative User Objects: How Microsoft Got a Good Idea Completely Wrong

GIAC Security Essentials Certification (GSEC) Practical Assignment Version 1.4b

Option 1 - Research on Topics in Information Security

Submitted: June 30, 2004

Table of Contents

Tables

1 Abstract

Windows 2000 and 2003 domains utilize a background process named the Security Descriptor Propagator process to enforce a specific level of security on the objects that Microsoft considers to be of an “administrative” level within the domain. Unfortunately, the current implementation of this process has many flaws that may actually hurt the ability of administrators to manage the security of a default windows environment. This paper attempts to explain the purpose and workings of this process, what the flaws are and how Microsoft may be able to fix them, and in some cases how administrators can get around these flaws in order to keep their environment secure.

2 Introduction and Background

I first came across the fact that Microsoft has a background process that runs on the PDC emulator to secure the object level permissions during the middle of 2003 when I was producing documentation for the security on a Windows 2000 Active Directory Forest that I had designed. While documenting the purpose of the default structures, I came across the AdminSDHolder container and did a search to find out what it was for. When I found out that it was the basis for further securing administrative objects, I was impressed with the fact that Microsoft was forward thinking enough to design a process such as this. I included the information about the purpose of the container within my documentation and didn't really think anything more about it. Unfortunately, this was only just the beginning for me.

Sometime at the beginning of 2004, I was called in to help with a specific issue that was occurring with Cisco's CallManager application and administrative users not being able to be assigned phones. Since our implementation of the application is Active Directory integrated and I knew that a service account needed rights to specific properties of the user object, I immediately knew that the issue related back to the permissions on the administrative user objects. Therefore, I suggested assigning read and write access to the user properties on the AdminSDHolder container would fix it.

Unfortunately, the solution that I proposed did not work and I ended up calling in Microsoft support in order to try to determine what the problem was. Before beginning with Microsoft, I gathered a lot of data surrounding the problem and sent it to them so that they could focus on the actual problem (which I knew to be related to the permissions for the protected user objects and the process around it) rather than trying to blame it on the CallManager application. Once we were on the phone the support person asked me to do a couple of other things to try to determine what the problem was and then she asked for time to go over the information that I had originally sent her. Fortunately, however, I also took a little time after we got off the phone and ended up determining the root cause of the problem based on the DSAcls outputs that I had provided earlier[1].

Once I had determined the root cause of the issue, Microsoft took the time to try to help me fix it even though their recommended fixes were not applicable

[1] The problem turned out to be that since I could not assign permissions directly to the correct property on the container object, I had instead assigned it to the "user objects." Since this only set an ACE flag to inherit the security to child objects rather than applying it to the object itself, the permission didn't give the service account the permission to write to the properties. The examination of the DSAcls output for the user object showed that the permission was replicated to the protected user object properly through the process, but the permission was only set to inherit to the user objects that were child objects of the protected user.

for this situation. This was my first real look at some of the quirks and flaws of the process and was when I started developing many of the possible workarounds for these flaws that are the basis for this whitepaper. Many of the statements about Microsoft support positions and feelings about the issues surrounding these flaws and workarounds that are included in this paper, but not directly attributable to a published source, are a direct result of these conversations about how to fix the issue that I was facing.

In the end, however, Microsoft and I failed to actually fix the problem through technical means because the solutions that I wanted to apply would not have been supported by Microsoft, and the solutions that they would support would have compromised the security of the network. Instead, the issue was solved by ignoring the technical problem itself and instead by obtaining a deviation to the security policy of the environment which stated that all users should only have a single user account. By adding a second non-protected account to the environment for these users, we were able to do a complete end-run around the issue without having to actually jump through hoops to fix it. Once this was solved, I put away my concern for it but all of the information that I had learned surrounding how the process works was stored away until this whitepaper was started.

The original intent of this whitepaper actually wasn't to solely deal with the issues surrounding how Microsoft protects administrative users. Instead, it was going to be an examination of how default security works within a Windows environment. But, when I got to the portion of the paper that dealt with protecting administrative users, I started digging deeper in order to be technically accurate and kept finding more and more flaws with how it is done. This went on until I realized that this section of the paper was longer and more interesting than anything leading up to it. So I decided to change the subject to specifically focus on this issue instead.

The original paper was not completely thrown out in conjunction with this change, however, as there are two holdover areas from the original paper still included in this version. These are the Object Level Security section and the information in the Default Object Level Security Settings appendix. Although this isn't specifically related to the problems that the whitepaper explores, it is something that I feel is critical to know in order to understand the issues involved. Therefore, I encourage you to take a look at both before diving into the issues section which represents the meat of the information.

With any luck, this introduction and background has given you some insight into where some un-attributed Microsoft statements come from, and why I know so much about a process that is pretty much hidden from the view of even some of the most knowledgeable network administrators. Also, I think it is important to point out that this paper is accurate as of its writing in June 2004 in regards to Windows 2000 domains running at the level of service pack 4, as well as Windows 2003 domains before the release of any service packs. According to my discussions with Microsoft, they are actively working

to change some portions of how the process that is described here works, and these changes will hopefully mitigate or even eliminate some of the issues and vulnerabilities discussed within this whitepaper.

In addition, the workarounds and suggestions within this paper are specific to fixing the problem that is being described within that section. Therefore, you may feel that a suggestion in one area may possibly conflict at some level with a suggestion in another. It should be understood that this whitepaper does not attempt to put together a cohesive design for dealing with all of the issues described, but rather attempts to provide suggestions about how to deal with each one individually based on the needs of your environment.

Finally, it may not show in this whitepaper because it is specifically about something that I have issues with, however I do have a lot of respect for Microsoft and the intelligence that its developers show in designing their systems. This is especially true in the areas of designing their systems to work with a very diverse computing base that ranges from the smallest offices to the largest corporations in the world for both the most technologically proficient support people and those that barely know how to turn on a computer[2]. However, because Microsoft's systems must be everything to everyone, I also feel very strongly that the default configuration is not necessarily the proper security context for an environment.

[2] Having worked with some of the largest corporations in the world, I have to say that sometimes I think the latter category is sometimes more applicable to the corporations.

3 Object Level Security

When an NTFS object is created in 2000/XP/2003 it is done with an empty security descriptor that allows inheritance, however when an Active Directory object is created it has explicit permissions set on the security descriptor for the object which may or may not allow inheritance based on the default security that is applied to that type of object during its creation. NTFS objects are therefore generally managed through inherited permissions, whereas Active Directory objects are managed by a combination of inherited and explicit permissions. This is a very important distinction to understand when thinking about the permissioning for Active Directory objects.

The explicit security settings for all objects within Active Directory (including all schema and site entries) are stored on the object itself inside the security descriptor (to be more technical, as if that is even possible in this paper, it is actually stored in the ntSecurityDescriptor attribute for the object). Microsoft defines a Security Descriptor in the following manner:

> *"A structure and associated data that contains the security information for a securable object."*[3]

The security descriptor for a securable object contains structural information about the security descriptor itself (size, offsets for different properties, etc…) and seven different properties: Revision; Resource Manager; ControlFlags; Owner; Group; Discretionary Access Control List (DACL); and System Access Control List (SACL). The contents of these properties determine which objects in the environment are allowed to access and manipulate the secured object, and whether that access is audited.

In addition to the explicit security stored within the security descriptor for an object, additional settings may be inherited from parent objects. Any settings that are inheritable by child objects are stored in the security descriptor of the parent object, and can be specifically set to inherit to all child objects or even to a certain type of child object (such as a user, group, or computer). Unless the explicit security of an object blocks the inheritance of settings, these inherited settings are used in conjunction with the explicit object level settings when the system assesses whether access should be granted or audited.

Whenever an object access is attempted, the system retrieves the token of the user or object attempting the access, then compares the token to the explicit settings contained within the security descriptor of the secured object. In Microsoft terms this is known as the lock (security descriptor) and key

[3] Definition of the term "security descriptor" from MSDN Platform SDK Security Glossary (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/s_gly.asp).

(token) mechanism that forms the basis of Windows access control[4]. If a match is found, the system acts according to the settings in the matched Access Control Entry (ACE) and either allows or denies access based on the matched ACE. If no match is found utilizing the explicit settings, then the system compares the token to the ACEs of the inherited permissions and either allows or denies access accordingly.

This is a very important, and often misunderstood, concept because of its subtlety. The ordering of permissions on objects in Windows 2000 and above isn't deny and then allow, it is deny and then allow explicit access, followed by deny and then allow inherited accesses in the order in which they are inherited by the object (i.e., deny then allow permissions inherited from the direct parent, followed by deny then allow permissions inherited from the grandparent, and so on through the entire tree of objects that are set to inherit to the securable object)[5].

Therefore, although inheritance allows you to delegate an object by adding to the permissions of the Active Directory object's DACL, it actually doesn't allow you to either remove or block a particular permission with a deny ACE at the container level or overwrite an inheritance block. This means that in order to lock down permissions to a level that is tighter than default for an object class you must either prevent a user from being able to traverse the container itself, or change the object level security on every object that you want to secure.

[4] For the purposes of this paper, we will be ignoring the "hacker using a credit card to pop open the lock because Microsoft didn't put a deadbolt in place by default" portion of the metaphor.

[5] See the "Order of ACEs in a DACL" reference in the MSDN Platform SDK Security (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/order_of_aces_in_a_dacl.asp) for details.

4 Security Descriptor Propagator Process

Microsoft recognized the fact that defining object level security for all objects of the same type would create an issue, because some objects should be protected to avoid the possibility of an escalation of privileges. Therefore, Windows programmers designed a specific process to protect the objects that are considered to be "administrative". The Security Descriptor Propagator (SDProp) background process runs on the Domain Controller that holds the PDC emulator FSMO role on an hourly basis, and is utilized to set the explicit permissions for each administrative user object. Administrative objects in the context of the SDProp process are those that have had the adminCount property of their object set to a value higher than zero.

Microsoft's stated purpose behind this process is to:

> *"(protect) administrative accounts from being modified by unauthorized users if the accounts are moved to a container or organizational unit in which a user has been delegated administrative privilege for the modification of user accounts."*[6]

In addition to preventing users that have rights over an OU from modifying administrative accounts that were mistakenly assigned into that OU, this process also protects administrative objects from being modified by members of the Account Operators group. By default, Account Operators are explicitly assigned full control over normal user and group objects. In contrast, the default security for administrative objects gives no specific rights to the Account Operators group[7].

The SDProp process first queries all groups in the domain that are set to be protected in order to determine all the objects that either have direct or transitive membership (membership in a group that is a part of the administrative group) and updates the value of the AdminCount property of the object to one. The process then queries administrative objects to determine if the security descriptor on the object matches the security descriptor on the AdminSDHolder container (CN=AdminSDHolder, CN=System, DC=*%Domain%*, DC=*%Root%*). If the descriptors match, the process moves to the next administrative object. If they do not match, the process replaces the security descriptor on the administrative object with a

[6] Quoted from knowledge base article 232199 - Description and Update of the Active Directory AdminSDHolder Object (http://support.microsoft.com/default.aspx?kbid=232199).

[7] For a full listing of default user and group object security descriptor settings and the default permissions on the AdminSDHolder see the Default Object Level Security Settings appendix of this document.

new descriptor that matches the settings of the AdminSDHolder container[8]. This is accomplished by directly copying the settings of the AdminSDHolder security descriptor over the security descriptor of the administrative object itself, however no other properties or values of the AdminSDHolder container are replicated to the administrative object that is being protected.

[8] The description of the operations of the SDProp process was created based information in the following knowledge base articles: 232199 - Description and Update of the Active Directory AdminSDHolder Object (http://support.microsoft.com/default.aspx?kbid=232199); 251343 - Manually Initializing the SD Propagator Thread to Evaluate Inherited Permissions for Objects in Active Directory (http://support.microsoft.com/default.aspx?scid=kb;EN-US;251343); and 318180 - AdminSDHolder Thread Affects Transitive Members of Distribution Groups (http://support.microsoft.com/default.aspx?scid=kb;en-us;318180) as well as testing and observation on a 2003 Domain Controller running in both 2000 Mixed and 2003 Native modes.

5 Issues with the Security Descriptor Propagator Process

As you have seen in the previous sections, the idea behind the SDProp process is necessary based on how Microsoft implemented security for Active Directory utilizing object level security. Unfortunately, the implementation of the SDProp process was not very well executed by Microsoft. The main purpose of this whitepaper is to discuss the issues in relation to this particular process that should be understood by a security administrator. Understand, however, that Microsoft generally defends their implementation of SDProp and therefore does not necessarily acknowledge these items as being issues or vulnerabilities that should be fixed in current versions of the OS. I also wish to point out that these issues and workarounds have been identified through working with Microsoft on the issue described in the introduction as well as testing and research on publicly available documents, however Microsoft has not provided me with any "inside" information or necessarily confirmed what I state in this section. Therefore, it is possible that my observations and/or research have led me to conclusions that are inaccurate and you should always confirm information for yourself before taking it as the absolute truth.

The issues that I have identified and are discussed below are as follows: the object utilized for the basis of the descriptor is a different object type than the objects that are being secured; the SDProp process will apply to any object type that has an adminCount attribute and is placed in a protected group; there is a vulnerability for some privilege escalation based on the default rights of the objects; the security applied through the process actually represents a loosening of security on the protected objects in some regards as opposed to the security on normal user objects; objects that are subjected to the SDProp process are "tattooed" with the permissions; and there is no supported way for an administrator to manage which groups are protected by the SDProp process.

5.1 Differing Object Type Issue

Unfortunately, because the designers of this process decided to utilize a container object as the basis for the security descriptor that is propagated, there is an irresolvable issue relating to assigning rights to attributes that are only available to the object class of the protected object, but not the container class to which the AdminSDHolder belongs. The best example of how this might manifest itself would be the issue that I faced with CallManager being an Active Directory integrated application that needs to

read from and write to the attributes that have been extended on the user object[9].

According to the Microsoft security philosophy, the right of an account (such as a service account) or group (possibly a group of users who manage the application) to write to this attribute would be assigned at the OU level and passed down to the normal user objects through inheritance. By default, however, the AdminSDHolder specifically blocks the inheritance of permissions in order to insure that the object level permissioning is not lessened by rights that might be inherited. Since the permission to write to the user property cannot be assigned directly to a container (because you cannot assign it to a container, unless the container class has also been extended), and the object is not set to inherit security, it is not possible to assign the permission to write to the specific attribute in a default configuration.

Microsoft can fix this issue by utilizing specific objects of the proper types (user, group, or computer) that would be stored within the AdminSDHolder container and that the SDProp process could reference to apply proper permissions. This would allow the assignment of security for the specific attributes that are valid for that object type rather than trying to utilize the permissions that are available on a container object as the basis for the SDProp process. Unfortunately, there is no good workaround that is supported by Microsoft for this problem currently.

One of Microsoft's official recommendations for dealing with this issue from a technical viewpoint is to change the default security on the AdminSDHolder so that it inherits the permissions from above rather than blocking that inheritance[10]. Unfortunately, this would specifically allow lower level users to manage objects which have higher level privileges (which is a security 101 level no-no). This suggestion can be made more palatable by actively managing the protected objects through placing them in their own "administrators" OU instead of storing them with all other non-protected user, group or computer objects (even if the other accounts or groups are also utilized by support rather than end-user personnel). If this is the case, you could block inheritance at the "administrators" OU level and explicitly set the permissions that are necessary at that level in conjunction with allowing inheritance.

[9] Read access is not an issue in a default configuration, because read access is given to the authenticated users group by the AdminSDHolder container security.

[10] See KB article 817433 - "Delegated permissions are not available and inheritance is automatically disabled" (http://support.microsoft.com/default.aspx?kbid=817433) for details.

The idea of a separate OU for "administrators" actually follows a Microsoft recommended practice for domain design[11]; however it does not actually address the stated purpose behind the process that is quoted above. In addition, the Microsoft recommendations surrounding the creation of an "administrators" account OU, or allowing inheritance as a solution to this problem, do not state that the security on these OUs should be changed from the defaults in order to accommodate the different security needs of these protected objects.

Another possibly less intrusive option (depending upon the accounts or groups involved) that is officially supported by Microsoft, would be to assign the write all properties permissions to the necessary account or group[12]. Although this would allow that particular user or group to manage administrative accounts, a security administrator may judge this to be less of a risk than giving permissions to the Account Operators or other group with full control over the OU in which the accounts reside (especially if we are only talking about a single service account).

The best option for dealing with this issue from a technical standpoint, however, is to add an ACE to the security descriptor of the AdminSDHolder container that is specific to the user or group that you want to allow to have access to the attribute. This could be achieved programmatically by directly adding an ACE with the proper information to the security descriptor. Unfortunately, when I proposed this to Microsoft as a workaround for my issue, they specifically stated to me that they would not support programmatically applying an ACE that isn't applicable to the security descriptor. The other way to achieve the same goal however, would be to change which attributes are available to the container class within the schema and then assign the specific rights that are necessary on the AdminSDHolder itself. Once the access control entry is assigned, the attribute could be unrelated from the object class with minimal (if any) technical issues[13].

This particular option was never part of my discussions with Microsoft personnel, so I cannot relate what their support position would be. But the

[11] See "The Best Practices OU Model" section of the "Best Practice Active Directory Design for Managing Windows Networks" document (http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/plan/bpaddsgn.mspx) for details.

[12] See KB article 817433 - "Delegated permissions are not available and inheritance is automatically disabled" (http://support.microsoft.com/default.aspx?kbid=817433) for details.

[13] I believe that a discussion of the technical ramifications of utilizing this option through a schema adjustment is very relevant to this whitepaper, however it also represents a serious tangent at this point because it is not directly related to the differing object type issue itself. Therefore, the technical ramifications of this option discussion is included as the "Discussion of Technical Issues Surrounding Adjusting the Schema" Appendix to this paper.

addition of an optional attribute to the object class is a standard method that is provided by Microsoft and therefore should be supported. What is questionable is whether the removal of this attribute and the leaving of a "dead" ACE entry in the security descriptor of the AdminSDHolder container object would be supported even though it causes no harm (it is the equivalent of the infamous "account unknown" that you would see if the ACE contained an entry for a SID of a deleted group or account). Although the Microsoft personnel that I was dealing with when I was discussing doing the same thing programmatically wouldn't go into details about why they wouldn't support that option, it is very conceivable that their discomfort with it had more to do with the fact that the security would be brute-forced into the security descriptor with a 3rd party program rather than the actual "dead" ACE that would be applied.

5.2 Application of Security Settings to non-User objects

Although the stated purpose for the SDProp process is to protect administrative level user objects, the SDProp applies to any object that is added to a protected group through either direct or nested membership and that the adminCount property is valid for. This means that objects such as computer accounts or groups that are mistakenly given membership in a group that is protected will also have their object level security re-written as a result. Since the Computer account has very different security needs than a user object, the security that is applied to the AdminSDHolder would have a seriously adverse effect on the computer account[14]. As for changing the security on a group, this change could create operational problems if the group isn't supposed to be limited to being managed by Domain Admin level personnel by default[15].

Microsoft can address this issue by changing the SDProp process to look at the type of object that the process is going to apply to, and then not run against any objects that are not specifically of the user or inetOrgPerson type[16]. Unfortunately, the only way to deal with this issue administratively at

[14] For a full listing of default computer object security descriptor settings and the default permissions on the AdminSDHolder see the Default Object Level Security Settings appendix of this document.

[15] For a full listing of default group object security descriptor settings and the default permissions on the AdminSDHolder see the Default Object Level Security Settings appendix of this document.

[16] inetOrgPerson is a new object type for Windows 2003 that is utilized for compatibility with X.500 directories. Since this object didn't exist in Windows 2000 and the security for the user and inetOrgPerson objects are exactly the same, I have chosen to utilize the user object to discuss almost all issues in this whitepaper except where I have felt that stating inetOrgPerson was necessary. However, the issues are exactly the same for all inetOrgPerson objects as they are for user objects in all cases described within the paper.

this moment is to make sure that computer or non-administrative group objects do not receive membership within a protected group, and to re-apply the default security settings for computer and group objects that have unfortunately been changed.

5.3 Possibility for Escalation of Privileges

The elimination of the escalation of privileges by non-authorized users is the basis of the idea behind the SDProp process, unfortunately it doesn't actually remove this possibility entirely. As you can see by the AdminSDHolder Container Default Settings table in the Default Object Level Security appendix, both the Administrators and Domain Admins groups have rights to manage the accounts and groups that the SDProp process applies to. However, this process also applies the security descriptor of the AdminSDHolder container to both the Enterprise Admin and Schema Admin groups (and users assigned to this group) in the forest root of the domain.

Therefore, by default, members of the Built-in Administrators group on the Domain Controllers and the Domain Admins of the forest root domain have the right to assign their own accounts into both the Enterprise Admins and Schema Admins groups and/or manipulate the accounts that have these group memberships assigned to them. That means that these users have the ability to escalate their own permissions from just managing the forest root domain to managing both the schema and the Active Directory structures in all domains within the forest.

According to my discussions with Microsoft personnel about this issue, they acknowledge this as a general issue but basically state that this is one of the main reasons you should be very careful when giving membership in the Domain Admins or Administrators groups of the forest root. Personally, I believe that is a valid answer in regards to mitigating the vulnerability (as is any alerting or reporting that would be done based on the addition of a user account to this group). However, from a security point of view, I believe that it should be considered to be a vulnerability and therefore should be addressed since there are many reasons for support personnel to have membership in these groups but not Enterprise Admins in an operational environment.

The second escalation of privileges threat is one that Microsoft and I specifically disagree on, and it relates to the privileges of the Administrators group itself. This particular issue is not limited to the SDProp process, but it is specifically applicable to it since we are talking about the management of the most powerful users within the network. In my opinion, the Administrators group for a Domain Controller (which is the Built-in Administrators group on the domain) represents a large violation of the "least privilege" security principle within the Windows environment as well

as providing no opportunity for a separation of duties between those who need administrative rights to the server and those who need to administer the logical infrastructure of the network.

In general, Microsoft believes that only those users who are Domain Admin level users or above should be given membership in this group because of the amount of privileges provided to this group[17]. However, this is not really feasible in a real-world operational environment. Many applications that are installed upon a server require service/application accounts that must be members of the administrators group in order to work properly[18]. If these applications provide services that are necessary on the Domain Controllers, then they must also be given membership in the Administrators group. Unfortunately, the amount of users who may be a member of the Built-in Administrators group, but don't require rights within Active Directory, is not always limited to the few service accounts necessary for applications on the Domain Controller.

In larger environments, different support teams are required to be utilized in order to manage many different aspects of the network. In one particular Windows 2000 world-wide enterprise environment that I have worked in, we limited the amount of accounts with Domain Admin group membership to around a dozen users by actively assigning the correct level of permissions for these support people to do their job at the OU level. However, in order to support the need for 24 x 7 support of the global infrastructure and the different operational teams that were responsible for it, the administrators group on the Domain Controller had around a hundred users assigned through different group memberships. Although I agree with Microsoft that this is way too many users with this level of access, each individual user with this membership had both a business case for why they needed it, and the political backing to override generic security concerns about too many users with Administrator access.

Since these accounts do not actually require any rights within Active Directory itself, the fact that they receive these incredibly powerful rights through their membership in this group is the basis for my feeling that this is a large violation of the principle of least privilege. This is especially true since there is no real security justification for why the Administrators group has privileges within Active Directory to begin with. The Domain Admins and Enterprise Admin groups are specifically assigned object level security permissions for every object within the domain, however the Administrators group is additionally given active directory rights at the root of the domain

[17] This includes some instances inside Active Directory that the administrators group actually has more rights on objects than the Domain Admins group itself.

[18] Microsoft is actively working to try to get vendors to write their applications so that this is not actually necessary, but it is a long way from actually happening at this point in time.

(and on some default objects that block inheritance) that are inherited down through the normal structures. The only difference in the effective rights of the Administrators and Domain Admins is that the Administrators are actually given "delete" permissions on some objects that the Domain Admins do not have assigned directly[19]. To me, this seems to reflect the "just make it work" mentality that is the traditional Microsoft style, rather than the new "secure by design" slogan that they say they are following today.

Although, it may seem like this whole discussion is tangential to the issue of possible escalation of privileges in relation to the SDProp process, I feel that this is very important for understanding whether the second escalation of privileges concern is valid or not. I have been told by Microsoft personnel in the Active Directory security area that "the BuiltIn- Admins group on DCs is the by far the most powerful group in the system[20]". If this is actually the case, then the fact that the Built-in Administrators group has the right to add themselves into the Domain Admins group (or the Enterprise Admins and Schema Admins if the Domain Controller is within the Forest Root) is not actually an escalation of privileges since these are lower level groups from the Built-in Administrators. However, I believe that each of these groups should be considered to have higher levels of authority than the Administrators group since I feel that they should only be utilized for managing the server itself rather than Active Directory. Therefore, from my point of view the default permissioning of the AdminSDHolder container that is replicated by the SDProp process does represent a second escalation of privileges vulnerability. In this case, it is up to the reader to decide for themselves about this issue.

There is an easy workaround for the Administrators group being able to escalate their privileges, but the dealing with the Domain Admins in the Forest root is much more difficult. Removing the Administrators group from the permissions of the AdminSDHolder in a domain would stop the Administrators group from being able to manage the membership and/or user objects that are protected by the SDProp process in that domain. This workaround could also work for the Domain Admins in the Forest Root. However it would require that you have Enterprise Admin level personnel

[19] The reason for this particular difference in rights is completely baffling to me since the Domain Admins are part of the Administrators group be default and therefore also receive these rights through this group membership.

[20] This quote is from a part of an e-mail from a Microsoft person regarding a discussion of the general subject of Active Directory security. However, because the writer was not aware that I might be quoting him, and this is not a publicly available quote, I do not feel it is proper to either identify him or provide a copy of the e-mail as a reference for this whitepaper. This is only provided as a reference to the fact that some people within Microsoft view the Built-in Administrators group in this way.

managing the protected objects within that domain which may cause an operational hardship.

5.4 Requirement for the Loosening of Security on Protected Objects

Although the purpose of the SDProp process is to specifically tighten security on the most powerful objects in the domain, not all of the security that is applied to normal user objects can be specified for the protected objects (see Differing Object Type Issue section of this paper for an explanation of why this is true). Therefore, the only way to assign the necessary accesses is to give the same level of access to all of the user attributes rather than specific ones. This issue is specifically relevant in regards to the Authenticated Users group having read access to all protected user objects rather than just the general information, public information, personal information, and web information property sets that is assigned by default on user objects.

Although read access is generally not considered to be an active vulnerability because it does not give you the ability to manipulate anything directly, it does represent access to information that could be utilized as part of the planning of an attack. Specifically, the fact that there is more information available to an attacker about an administrative level user object than a normal one is a cause for alarm. In addition, the ability to read different information on "protected" user objects versus a normal user object gives an attacker a way to identify these protected accounts[21].

User objects are the only example that I know of within the Active Directory environment where read access is limited to specific attributes rather than the entire object. I would guess that this heightened level of security for user objects would specifically mean that Microsoft felt that allowing read access to the entire user object represented a security concern. The basis for this concern may be that providing too much information about user objects to anyone who is authenticated by the domain (or any domain that is trusted by the domain) may leave users vulnerable to social engineering. Therefore, the developers may have specifically designed the security so that authenticated users would not have read access to all of the information contained in the user object[22]. However, this extra precaution is

[21] There are actually much easier ways to figure out high value targets in the default security environment for Active Directory, such as querying Domain Admins for its membership since authenticated users have read access to the all properties of both the normal and protected groups by default, but these design failures are the subject for a completely different paper.

[22] This is pure speculation as to their motives for the tighter security in this area, I have no specific reference or confirmation from Microsoft for these statements but it is the only motive that I can come up with for why this was done.

absent for the most important user objects in the domain. Because the base reason for this vulnerability is the same as the Differing Object Type issue, the possible solutions and workarounds for this issue are detailed in that section.

5.5 Tattooing of Permissions on Protected Objects

The SDProp process is a standalone mechanism that permanently changes both the adminCount property and the security of the object that it is applied to. The SDProp process also keeps no internal record of the changes that it makes. Therefore, once the SDProp process has been run against an object because it is a member of a protected group, there is no way to reverse either the adminCount value or the security descriptor that was overwritten within the SDProp process itself. Permanently changing the security of an object through a process that cannot be reversed by that process is referred to as "tattooing" the object security[23].

This tattooing of permissions has important ongoing support implications for the administrators of a network since the security will remain in place even if the object is no longer part of a protected group. In addition, if the SDProp process no longer applies to the object, but the security on the object has not been reversed or inheritance enabled, then neither changes to the security at the OU level nor changes to the AdminSDHolder will be applied to the object.

Microsoft recognizes this problem and provides a "solution" for it through a script that can be run against the domain that resets the adminCount property to zero for all user objects they have a value of one when the script is run[24]. Additionally, this script re-enables inheritance on these user objects in order to allow management from an OU rather than just at the object level. Unfortunately, the script stops short of actually resetting the object level permissions so this solution does not address the fact that the tattooing has occurred within the object level security.

Microsoft may not consider the fact that the permissioning on the object is still that of the AdminSDHolder container to be an issue; however it definitely could be from a security management and operational standpoint. Remember that the stated purpose of the SDProp process is to limit who can manage the object. This is a good thing in order to prevent an escalation of privileges, however in real world terms this means that only a

[23] This term is also utilized in the Microsoft lexicon when discussing applying certain permissions onto objects through GPO such as NTFS, Registry Key, or Service objects.

[24] See KB article 817433 - "Delegated permissions are not available and inheritance is automatically disabled" (http://support.microsoft.com/default.aspx?kbid=817433) for details on the script.

limited number of people can manage these accounts. If a user is no longer part of the protected group, and therefore there is no longer a risk for escalation of privileges, it should be managed by the normal security processes of the environment in order to limit the total cost of operations (TCO) of the network. In addition, the script is only applicable to user objects rather than all types of objects that may have had their security replaced by the SDProp process.

Since the object level permissions are not reversed by the script, any environment that is following Microsoft recommendations (relying on default mechanisms and groups) will not actually have the rights that are necessary. This is true in relation to the Account Operators group not having the right to manage the user since its rights come from the default object level security rather than being inherited from the Active Directory structures. In addition, the "RAS and IAS Servers" group is given specific object level permissions through the default user object security in order to authenticate domain users who remotely access the network, but these permissions are not assigned to the AdminSDHolder for application to protected users. Therefore, a user account that has previously been a protected object does not have the ability to be authenticated through an IAS server for remote access to the network even after the script provided by Microsoft has been run.

Microsoft can correct this issue by changing the SDProp process (or creating a different process entirely) so that it actually queries to see if an account has an adminCount that is higher than zero, but is not currently part of a protected group. When it finds one it could reset the adminCount property to zero on the object and temporarily hold the names of the accounts that it changed in memory. Once the querying portion of the SDProp process is completed, the process could then utilize the value of the defaultSecurityDescriptor for the object type to re-assign default object level security to the objects that were stored in memory. This would make the SDProp process totally reversible for user and computer objects and completely remove this issue for all but the mistakenly protected group objects.

As a current workaround for correcting this problem with user objects, I would propose utilizing the script to identify all of the users that need to be reversed (the script includes pop-ups showing the names of the user accounts that are having their settings changed) and then changing the object level security on those user objects back to the default[25]. In a Windows 2003 environment, this can be done by clicking the "Default" button on the "Advanced Security Settings" screen for the user object itself.

[25] For a full listing of default user object security descriptor settings and the default permissions on the AdminSDHolder see the Default Object Level Security Settings appendix of this document.

In a Windows 2000 environment, this would either have to be done by hand, through a tool (such as DSAcls), or through a specially written script.

5.6 Lack of Control over SDProp Process

While I agree that the idea behind the creation of the SDProp Process is a very good one given the way that default security within Windows works, I also believe that there is a large difference between specifying a default configuration that can be changed to meet the specific needs of an environment and building in a process that changes basic security functionality based on group membership. Unfortunately, the SDProp process does the latter and this can create havoc within a network at times. To make matters worse some of this havoc has been created by Microsoft itself because of how it has constantly changed the SDProp process.

According to Microsoft documentation, the scope of the SDProp process has been continually altered since it was first designed. During the release candidate phase of Windows 2000, the SDProp process only applied to the Domain Admins and Administrators groups[26]. This was apparently changed when Windows 2000 was released to include the Enterprise Admins and Schema Admins as well as the original Administrators and Domain Admins member user and group objects.

Microsoft later extended the scope of the process without any real notice[27] to include the following groups as well as the four listed above: Account Operators; Backup Operators; Cert Publishers; Domain Controllers; Print Operators; Replicator; and Server Operators[28]. The process also specifically controls the security on the default domain administrator account as well as the "Krbtgt" account, however it is unclear from the documentation whether this change was also imposed as part of a change in scope or if it always existed. The groups listed above also reflect the groups managed by the SDProp process in Windows 2003.

The changing of the basic security structure of Active Directory without notice doesn't really constitute a vulnerability (especially when the change actually tightened the security) or is specifically limited to the SDProp

[26] Listing from KB article 232199 – "Description and Update of the Active Directory AdminSDHolder Object" (http://support.microsoft.com/default.aspx?kbid=232199).

[27] Microsoft did this with no notice or documentation about the change to the SDProp process in a hotfix for an issue surrounding MaxTokenSize (KB 327825). This hotfix was included in the release of SP4 and very little notice or documentation was provided to explain the implications of this upgrade to customers at that point either, however searching at that point would allow you to find specific KB articles explaining the issues after they occurred.

[28] Listing from KB article 817433 – "Delegated permissions are not available and inheritance is automatically disabled" (http://support.microsoft.com/default.aspx?kbid=817433).

process. However, changing basic object level security in Active Directory with very little notice of the change does constitute a very important general issue in my opinion and does exacerbate the "Lack of Control" issue with the SDProp and other processes. This larger issue actually also applies to other changes that were made with little mention in SP4 such as how restricted group policy worked before and after the upgrade.

The reason why the "Lack of Control" issue is applicable to the SDProp process actually isn't this larger issue of notice, it's that Microsoft decided to specify what will be managed by the SDProp process in the first place rather than giving administrators the ability to choose how and what this process applies to. I have discussed this issue with Microsoft personnel and apparently the discontent over this particular issue is widespread since the expansion of scope occurred with the introduction of Windows 2000 SP4. According to my conversations, Microsoft is currently working on a fix for this particular issue that would allow security administrators to choose which particular groups the SDProp process applies to. However, it is unclear whether Microsoft will be providing the ability to manage all of the groups that the SDProp process will apply to; give administrators the ability to manage all groups that they wish to through SDProp (including customer created groups with "administrative" rights, but not membership in the default groups that are currently protected); or only the groups that were newly included in the latest expansion of scope.

My suggestion for dealing with this issue is more radical than what Microsoft will probably be providing as a solution (although it has been passed to Microsoft personnel). This suggestion would be to provide security administrators with the ability to completely manage the groups that the SDProp process applies to by allowing the creation of "AdminSDHolder" objects with specific object level security settings for different adminCount levels (if a user object was a member of multiple protected groups with different adminCount values, then the lowest/highest setting would be applied to the adminCount for that user and the appropriate source object would be utilized for specifying the security). Doing this would allow an administrator to specify differing object level security for any and all groups that they wished or just leave it at the default settings that Microsoft currently is providing as a requirement. Currently, however, there is no workaround that will allow you to specify which groups are managed by the SDProp process that I know of.

6 Appendix A - Default Object Level Security Settings

Because of how Windows manages explicit versus inherited security, it is important to understand the default object level security settings for the different object types in order to understand the issues surrounding the SDProp process. Although administrators are allowed to explicitly define security on an object, it is a security management best practice that administrators utilize containers or OUs to manage security rather than trying to do so through the objects themselves. Therefore, the security that is placed on an object at creation is generally the only explicit permissions that are set and they cannot be overridden from above[29].

The tables in the following sub-sections describe the security that is placed directly on the objects and object types that are relevant to understanding the SDProp issues. There are a couple of caveats to keep in mind in regards to the tables of default rights:

The permissions listed in the tables only represent the permissions that are explicitly set on an object during creation. However, some additional permissions are assumed to be applicable to the objects that allow inheritance by Microsoft and other vendors because they are set to inherit to all child objects from the root of the domain. These would include (but are not limited to) permissions for the Administrators, Enterprise Admins, and Pre-Windows 2000 Compatible Access groups on standard user, group and computer objects.

Unless inheritance to other objects is specifically mentioned the rights assigned by the default security only applies to the object itself and is not inherited by any child objects that may be contained within the object type specified.

Full control permissions in the listings below actually refer to setting all of the bits that are currently assigned value within the security descriptor of active directory objects. This is not to be confused with the “full access” setting that can be conferred through the SDDL, which actually sets a special bit inside the descriptor. The term full control is utilized to describe the access rights for different groups because there is effectively no difference between the actual settings applied and “full access”.

[29] Explicit access for an object can be superceded at a higher level by denying a user the ability to traverse the path to the object (denying full control to a user for an OU that contains the object). However, this is part of the parent object’s security (rather than the object itself) and cannot be utilized if that user needs access to any part of the object.

The "Read Control" permission refers to the ability to read the DACL, group and owner properties of the security descriptor but not the SACL.[30] This means that a user with this right is allowed to view the security settings, the owner and the primary group that are assigned to the object, but not the auditing settings.

6.1 Computer Objects

Table 1 – Computer Object Default Settings

Assigned to:	Permissions
Domain Admins	Full Control
Account Operators	Full Control
System	Full Control
Authenticated Users	Read (specifically read all properties, list contents, list object, and read control)
Cert Publishers	Read/Write userCertificate property
Creator Owner	Self for service principal name and DNS host name attributes; Write user account restrictions property; Control Access to all extended properties; Read (specifically read all properties, list contents, list object, and read control); Delete; and Delete Tree
Everyone	Control Access for user-change-password property
Print Operators	Create/Delete Printers
Self	Self for service principal name and DNS host name attributes; Read/Write to values stored in the Personal Information property set; and Create Child/Delete Child for all objects
Windows Authorization Access Group	Read tokenGroupsGlobalAndUniversal property (Windows 2003 Only)

[30] This information comes from the Read_Control definition on the standard access rights page of the MSDN Platform SDK Security (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/standard_access_rights.asp).

6.2 Group Objects

Table 2 – Group Object Default Settings

Assigned to:	Permissions
Domain Admins	Full Control
Account Operators	Full Control
System	Full Control
Authenticated Users	Read (specifically read all properties, list contents, list object, and read control); Control Access to send-to property
Self	Read (specifically read all properties, list contents, list object, and read control)
Windows Authorization Access Group	Read tokenGroupsGlobalAndUniversal property (Windows 2003 Only)

6.3 User Objects

Table 3 – User Object Default Settings

Assigned to:	Permissions
Domain Admins	Full Control
Account Operators	Full Control
System	Full Control
Authenticated Users	Read Control; Read for general information, public information, personal information, and web information property sets
Cert Publishers	Read/Write userCertificate property
Everyone	Control Access for user-change-password property
RAS and IAS Servers	Read for membership, RAS information, user account restrictions, and user logon property sets
Self	Read (specifically read all properties, list contents, list object, and read control); Control Access for receive as, send as, and user change password properties; Read/Write for the email information and web information property sets

Assigned to:	Permissions
Windows Authorization Access Group	Read tokenGroupsGlobalAndUniversal property (Windows 2003 Only)
Terminal Server License Servers	Read/Write terminalServer property (Windows 2003 Only)

6.4 Default AdminSDHolder Security Settings

Inheritance is blocked on the AdminSDHolder object, so the following explicit settings are the only security that is applicable to the administrative user objects. For this reason, some items are explicitly set on this object whereas they are normally inherited from the domain security.

Table 4 – AdminSDHolder Container Default Settings

Assigned to:	Permissions
System	Full Control
Administrators	Full Control minus delete
Domain Admins	Full Control minus delete and delete subtree
Enterprise Admins	Full Control minus delete and delete subtree
Authenticated Users	Read (specifically read all properties, list contents, list object, and read control)
Everyone	Control Access for user-change-password property
Personal Self	Control Access for user-change-password property
Cert Publishers	Read/Write userCertificate property
Windows Authorization Access Group	Read tokenGroupsGlobalAndUniversal property (Windows 2003 Only)

Assigned to:	Permissions
Terminal Server License Servers	Read/Write terminalServer property (Windows 2003 Only)
Pre-Windows 2000 Compatible Access	These permissions are inherited to all user and inetOrgPerson objects that are children of the object[31]. Read (specifically read all properties, list contents, list object, and read control) and Read Property for the remote access information, general information, group membership, account restrictions, and logon information property sets.

[31] These permissions are effective for any user or inetOrgPerson objects that are stored within the AdminSDHolder container itself, but are non-effective when applied directly to user or group objects. See the Differing Object Type Issue section of this paper for a full explanation. Also, the inetOrgPerson object type is only available in Windows 2003 and later domains.

7 Appendix B - Discussion of Technical Issues Surrounding Adjusting the Schema

I want to note that this appendix is not a technical discussion of all issues surrounding the schema, but rather it is only a discussion of the ramifications of utilizing a temporary schema adjustment to allow the assignment of permissions to attributes on the AdminSDHolder container. Regrettably, Microsoft has so thoroughly convinced people of the power and danger of the schema that even the mere mention of editing or changing values within it sends otherwise thoughtful administrators running in panic. Sadly, I am old enough to remember way back in the days of NT 4.0 when the thought of editing the registry did the exact same thing (mostly because Microsoft constantly screamed that editing the registry could cause system wide issues). This does not mean you shouldn't have a healthy amount of respect for the schema or that you should randomly change things without knowing what you are doing, however the thought of utilizing the schema to improve your environment should not be something that is immediately rejected out of blind fear and ignorance.

One of the biggest misconceptions about the schema is that you cannot reverse any changes you make to it. This is based on the fact that you cannot remove any schema extensions, but this is very different from the schema adjustment that I am referring to in this paper. An extension is the addition of a new attribute or object type to the schema partition. Once this addition is made to the environment, Microsoft will not allow you to remove these extensions because there is no way to know whether any objects are actually utilizing and/or storing information about the extended attributes or objects within Active Directory. Since leaving this information within Active Directory is the equivalent of having some old and useless information within a database (we all know that would never happen), and removing the attribute or object type does constitute a risk, Microsoft believes that it just isn't worth it to remove this extra information and I agree with them.

Although most people utilize the terms schema modification, change, or adjustment interchangeably with schema extension, you should realize that very few changes to the schema actually represent an extension. This is specifically the case in the adjustment that I proposed as part of the workaround to the Differing Object Type issue. This adjustment would temporarily relate the attribute to the container class. This relation would allow you to assign permissions for a user or group to have read or write access to the property on the AdminSDHolder, but has almost no real implications from an Active Directory perspective.

A schema extension, on the other hand, can have performance and storage implications. This is not because of what the extension actually does to the

data table within the NTDS.dit file that acts as the storage mechanism for the Active Directory database, but rather because of the additional data that could be stored there. Microsoft describes how the data table works in the following way:

> *"The data table can be thought of as having rows (each representing an instance of an object, such as a user) and columns (each representing an attribute in the schema, such as GivenName). For each attribute in the schema, the table contains a column, also called a field. Field sizes can be fixed or variable. Fixed-size fields contain an integer or long integer as data type; variable-size fields typically hold string types (for example, Unicode strings). The database allocates only as much space as a variable-size field needs: 16 bits for a 1-character Unicode string, 160 bits for a 10-character Unicode string, and so on."*[32]

Therefore, the extension of the schema to add a new attribute to the environment does not necessarily grow the database very much, but it does allow the data table (and thus the NTDS.dit database) to grow exponentially if the new attribute is utilized to store data. Nevertheless, this new field within the data table is created during the extension and is only utilized when data is stored in the attribute for an object.

Usually, an adjustment to the schema consisting of relating an optional attribute to an existing object class also would have this same kind of impact. The relation of an existing attribute to a class actually occurs within the schema table of the NTDS.dit database, rather than the data table, and consists of a single additional entry in the object class' "row". However, the implications of making this kind of change on a permanent basis would be that the attribute would be available for storage of data for all objects of that class, and the storage of that data would grow the data table in the same way that the original extension would have.

Fortunately, the temporary schema adjustment that is relevant to this issue would not require the storage of any data against the related attribute of the AdminSDHolder or any other container object. This adjustment would only be utilized to assign the permission to the AdminSDHolder, and then the attribute would be unrelated from the class thus allowing no storage of information against that attribute to occur. Therefore, the only actual impact to the NTDS.dit file during this procedure would be a temporary increase in the size of the schema table for the assignment of the optional attribute to the class, and an increase in the data table equivalent to the size of a single ACE being added to the ntSecurityDescriptor attribute for the AdminSDHolder object.

---

[32] Quoted from "Optimizing Size Requirements for Growth in Directory Service" (http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/maintain/adsize.mspx).

8 Appendix C – How to Assign Permissions to User Attributes Through the AdminSDHolder

The following are the step-by-step instructions for assigning permissions for an account or group to write to a specific attribute on the AdminSDHolder container so that these permissions can be replicated to the user objects protected by the SDProp process. Before beginning, the following must be true:

- The User must be utilizing the server that holds the Schema Master FSMO Role for the Forest (this is not necessarily a requirement for editing the schema, but the steps assume that the operation is being done locally).
- The Active Directory Schema MMC must be available (this requires that the schmmgmt.dll is registered as a service on the server, see knowledge base article 285172 for detailed instructions)
- The ability for the schema to be updated from the server must be turned on (see knowledge base article 285172 for detailed instructions)
- The User must be a member of the Schema Admins group

8.1 Step 1 – Relating the Attribute to the Container Class

Utilize the following steps to relate the necessary attribute to the container class:

1. Open the Active Directory Schema MMC by doing the following:
   a. Start an MMC from the start menu by choosing Start → Run and typing "MMC" in the command box
   b. Select the CTRL + M key combination
   c. Click the "Add" button in the "Add/Remove Snap-in" window
   d. Double-click the "Active Directory Schema" entry in the "Add Standalone Snap-ins" window
   e. Click the "Close" button in the "Add Standalone Snap-ins" window
   f. Click "OK" in the "Add/Remove Snap-in" window
2. Navigate to the container class in the left hand pane by expanding "Active Directory Schema [*%Domain%*]" and "Classes"
3. Right-click "container" in the left-hand pane and choose "Properties" from the pop-up menu
4. Select the "Attributes" tab

5. Click the “Add…” button next to the Optional window
6. Within the “Select Schema Object” window, select the attribute that you wish to secure through the AdminSDHolder container
7. Click the “OK” button in the “Select Schema Object” window
8. Click “Apply” and confirm that the attribute that was selected appears in the “Optional” attribute window

> ✉ Repeat the previous six instructions to add additional attributes to the container class

9. Click “OK” to close the “container Properties” window
10. Close the MMC (I suggest saving it as this point, because you will need it again later)

8.2 Step 2 – Make Attribute Viewable

Utilize the following steps to make the new attribute available through the Active Directory Users and Computers MMC:

1. Open the dssec.dat[33] file utilizing notepad by choosing Start → Run, typing “dssec.dat” in the command box, and selecting notepad from the available programs (if necessary). If this does not work because the path statement has been modified use the following procedure:
   a. Open Windows Explorer by right-clicking on the start menu and choosing “Explore”
   b. Navigate to the *%SystemRoot%* and choose the “System32” directory in the left hand pane

   > ✉ By default the *%SystemRoot%* in Windows 2000 is the C:\Winnt directory, in Windows 2003 it is C:\Windows

   c. Double-click the “dssec.dat” file and select notepad from the available programs (if necessary)
2. Save the original file as dssec.bak by using the “File” → “Save as…” command before continuing
3. Search for the user section of the data file, it should begin like the following string:

[33] For information about the role of the dssec.dat file and how to manipulate the viewable attributes for the Active Directory Users and Computers MMC see webpage #3 from the “Security Active Directory” chapter of the Inside Active Directory: A System Administrator's Guide available online at the Windows IT Library website (http://www.windowsitlibrary.com/Content/667/04/3.html).

[user]

4. Find the attribute that you assigned to the container class in Step1 and copy the string to memory by highlighting it and using the CTRL + C key combination
5. Search for the container section of the data file, it should begin like the following string:

   [container]

6. Under the container section, paste the attribute string to the proper place in alphabetical order making sure that each attribute is on its own line inside the file
7. If necessary, change the value that the attribute is equal to from "7" to "0" so that it appears as the following: *%Attribute%*=0
8. Save the edited file as dssec.dat by using the File → Save as… command
9. Close and then Re-open the dssec.dat file to confirm that your edit saved properly.

8.3 Step 3 – Reboot

In order for both the change in the container class and the edit of the dssec.dat file to be incorporated into memory, the server must be rebooted before continuing.

8.4 Step 4 – Assign Permissions

Once the server has rebooted and you have logged on with rights that would allow you to edit the security on the AdminSDHolder container it is possible to utilize one of the two following methods to assign the permissions. If you are only adding a single ACE, I recommend utilizing the first method. However, the second is more useful if you have many changes that need to be made; you need to specify these changes for someone else to apply; or you want some type of repeatable action that can be tested in a lab and then applied to different environments.

8.4.1 Active Directory Users and Computers Method

Utilize the following steps to assign the permissions to the attribute through the MMC:

1. Open the Active Directory Users and Computers MMC

> These steps require that the MMC has “advanced features” enabled. If there is not a checkmark next to “advanced features” in the view menu of the MMC then select this option before continuing.

2. Navigate to the AdminSDHolder container in the left hand pane by expanding “Active Directory Users and Computers [*%DomainController%*]”; “*%Domain%*”; and “System”
3. Right-click the AdminSDHolder container and choose “Properties” from the pop-up window
4. Select the “Security” tab in the “AdminSDHolder Properties” window
5. Click the “Advanced” button within the “Security” tab
6. Click the “Add” button on the “Permissions” tab of the “Advanced Security Settings for AdminSDHolder” window
7. Type or select the name of the user or group that you wish to have access to the user attribute and click “OK” to bring up the “Permission Entry for AdminSDHolder” window
8. Select the “Properties” tab in the “Permission Entry for AdminSDHolder” window
9. In the “Permissions” control area of the “Permission Entry for AdminSDHolder” window, find the attribute that you wish to assign permissions for and check the appropriate boxes

> If the attribute does not show in the window, and you are setting this permission for a domain other than the forest root, then you may have to wait for the global catalog to replicate the schema change before being able to complete the previous step. If you are doing this for the forest root, or you have waited long enough for the Global Catalog to replicate, then go back to Steps 1 and 2 of the procedure to make sure that the changes you made are still intact.

10. Click “OK” and check the “Permissions Entry” window to make sure that the permission was assigned correctly
11. Click “Apply” on the “Advanced Security Settings for AdminSDHolder” window to make the change permanent

> Repeat the previous six instructions to add permissions for additional groups or users.

12. Close the “Advanced Security Settings for AdminSDHolder” window and insure that all settings are applied by clicking “OK”

13. Close the “AdminSDHolder Properties” window and insure that all settings are applied by clicking “OK”

8.4.2 DSAcls Method

DSAcls[34] is a very powerful command line tool that is available as part of the support tools for both Windows 2000 and Windows 2003 Servers. In order for you to use these instructions, the support tools must be loaded on the server that you are making the changes from. Also, a simple batch script with multiple commands can be easily created to cycle through the setting of multiple ACEs. The following command should be used to assign a permission to the attribute through the command prompt or script (the specific separators in the command must be exact in order for the command to work):

dsacls "*%Path%*" /G "*%Domain%**%SecurityPrincipal%*:RPWP;*%Attribute%*"

Table 5 – DSAcls Key

Variable	Explanation
dsacls	The command for calling the dsacls.exe program from the support tools folder.
%Path%	The full path to the object. An example of a path for the “test.domain.com” domain would be “CN=AdminSDHolder, CN=System, DC=test, DC=domain, DC=com”.
/G	Grant Access
%Domain%	The NT style name of the domain. An example would be “Test” rather than “test.domain.com”.
%SecurityPrincipal%	The name of the user or group that you want to provide access to. An example would be “Backup Operators”.
RP	Read Property
WP	Write Property
%Attribute%	The name of the attribute that you are specifically trying to add the access to.

[34] The information about DSAcls provided here is specific for this process, however DSAcls is a general tool that can be utilized to secure any object within active directory through a script or from a command line. For more information, see knowledge base article 281146 - How to Use Dsacls.exe in Windows 2000 (http://support.microsoft.com/default.aspx?scid=kb;en-us;281146) or use the “DSAcls /?” command for the help menu.

8.5 Step 5 – Wait for SDProp Process to Replicate Security

The SDProp process replicates the security from the AdminSDHolder to the protected user objects on an hourly basis. According to knowledge base article 251343 - Manually Initializing the SD Propagator Thread to Evaluate Inherited Permissions for Objects in Active Directory (http://support.microsoft.com/default.aspx?scid=kb;en-us;251343) it is possible to initiate the SDProp process in a Windows 2000 environment, however I have not been successful utilizing the instructions in this article against Windows 2003. In either case, you should confirm that the new ACE is applied to a protected object (such as the default domain administrator account) before declaring that the assignment of rights was successful.

8.6 Step 6 – Unrelating the Attribute from the Container Class

Utilize the following steps to unrelate the attribute from the container class:

1. Open the Active Directory Schema MMC that you saved earlier or go back to step one and follow the instructions to create an Active Directory Schema MMC.
2. Navigate to the container class in the left hand pane by expanding "Active Directory Schema [*%Domain%*]" and "Classes"
3. Right-click "container" in the left-hand pane and choose "Properties" from the pop-up menu
4. Select the "Attributes" tab
5. In the Optional window of the "Attributes" tab, select the attribute that was added to the container class earlier and click the "Remove" button (repeat for all attributes that were added in step 1)
6. Click "Apply" and confirm that the attribute that was selected no longer appears in the "Optional" attribute window
7. Click "OK" to close the "container Properties" window
8. Close the MMC

8.7 Step 7 – Reboot

In order for both the change in the container class to be incorporated into memory, the server must be rebooted. It is not necessary to change the

dssec.dat file back to its original form because the attribute will not show in the MMC if it is not related to the object type. In addition, leaving the dssec.dat file as it is allows you to skip that step if you ever have to change the attribute security again. Once the server has rebooted, the process is completed

9 References Section

The following sources were utilized for both background and specific references within this paper:

"Best Practice Active Directory Design for Managing Windows Networks: The Best Practices OU Model." 2004. URL: http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/plan/bpaddsgn.mspx (29 June 2004).

"Knowledge Base Article 232199: Description and Update of the Active Directory AdminSDHolder Object." 8 April 2004. URL: http://support.microsoft.com/default.aspx?kbid=232199 (14 May 2004).

"Knowledge Base Article 251343: Manually Initializing the SD Propagator Thread to Evaluate Inherited Permissions for Objects in Active Directory." 24 Sept. 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;EN-US;251343 (14 May 2004).

"Knowledge Base Article 281146: How to Use Dsacls.exe in Windows 2000." 18 Dec. 2003. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;281146 (14 May 2004).

"Knowledge Base Article 318180: AdminSDHolder Thread Affects Transitive Members of Distribution Groups." 8 April 2004. URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;318180 (14 May 2004).

"Knowledge Base Article 327825: New Resolution for Problems That Occur When Users Belong to Many Groups." 26 Sept. 2003. URL: http://support.microsoft.com/default.aspx?kbid=327825 (14 May 2004).

"Knowledge Base Article 817433: Delegated Permissions are not Available and Inheritance is Automatically Disabled." 20 May 2004. URL: http://support.microsoft.com/default.aspx?kbid=817433 (29 May 2004).

Kouti, Sakari, and Mika Seitsonen. "Securing Active Directory." Inside Active Directory: A System Administrator's Guide. Boston: Addison Wesley Professional, 2001. Windows IT Library. URL: http://www.windowsitlibrary.com/Content/667/04/3.html (29 June 2004).

"MSDN Platform SDK Security: Order of ACEs in a DACL." May 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/order_of_aces_in_a_dacl.asp (20 May 2004).

"MSDN Platform SDK Security: Read_Control Definition." Feb. 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/standard_access_rights.asp (7 March 2004).

"MSDN Platform SDK Security: Security Glossary." Feb. 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/s_gly.asp (7 March 2004).

"Optimizing Size Requirements for Growth in Directory Service." 2004. URL: http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/maintain/adsize.mspx (29 June 2004).