Realtime publishers

The Definitive Guide™ To

Active Directory Troubleshooting, Auditing, and Best Practices

2011 Edition

Don Jones

Copyright Statement

Chapter 4: Active Directory Security

In the security world, *AAA* is usually the term used to describe the broad functionality of security: authentication, authorization, and auditing. For a Windows-centric network, Active Directory (AD) serves one of those roles: authentication. Internally, AD also has authorization and auditing functionality, which are used to secure and monitor objects listed within the directory itself. In this chapter, we'll talk about all of these functions, how AD implements them, and some of the pros and cons of AD's security model. We'll also look at reasons your own security design might be due for a review, and potentially a remodel.

This chapter will also discuss security capabilities usually acquired from third parties. I know, it would be nice to think that AD is completely self-contained and capable of doing everything we need from a security perspective. In a modern business world, however, that's rarely true, as we shall see.

Active Directory Security Architecture

As mentioned, AD has a role in each of the three main security functions. Let's take each one separately.

Authentication: Kerberos

Microsoft adopted an extended version of the industry-standard Kerberos protocol for use within AD. Compared with Microsoft's older authentication protocol, NTLM, Kerberos provides distinct benefits:

- Mutual authentication. Both sides of any security transaction are identified and authenticated *to each other.* With NTLM, the client was authenticated, but the client wasn't able to verify the server's identity.
- Distributed processing. Clients are responsible for maintaining 100% of the information needed to authenticate themselves to a server; servers maintain nothing. That behavior reduces server overhead, improving overall performance.
- Secure. Unlike NTLM, Kerberos doesn't transmit any portion of your password over the network at any time—not even in encrypted form. Thus, passwords remain a bit safer.

The name *Kerberos* comes from Greek mythology, and identifies the mythical three-headed dog that guarded the gates to the Underworld. The *three-headed* bit is the important one because the protocol entails three parties: the client, the server, and the Key Distribution Center (KDC).

In AD, Kerberos relies on the fact that the KDCs—a role played by domain controllers—have access to a hashed version of every user and computer password. The users and computers, of course, know their passwords, and the computers (which users log on to, of course) know the same password-hashing algorithm as the domain controllers. This setup enables the hashed passwords to be used as a symmetric encryption key: If the KDC encrypts something with a user or computer password as the encryption key, that user or computer will be able to decrypt it using the same hashed password.

When a user logs on, their computer—on the user's behalf—contacts the KDC and sends an authentication packet. The KDC attempts to decrypt it using the user's hashed password, and if that is successful, the KDC can read the authentication packet. The KDC constructs a *ticket-granting ticket (TGT),* encrypting it first with its *own* encryption key (which the user doesn't know), then again with the *user's* key (which the user does know). The user's computer stores this TGT in a special area of memory that isn't swapped to disk at any time, so the TGT is never permanently stored. The TGT contains the user's security token, listing all of the security identifiers (SIDs) for the user and whatever groups they belong to.

When the user needs to access a server, their computer resends the TGT to a domain controller. The domain controller decrypts the TGT using its private key—keep in mind that there's no way the user could have tampered with the TGT and still have that decryption work because the user doesn't have access to the domain controller's private key. The KDC creates a copy of the TGT called a *ticket,* and encrypts it using the hashed password of whatever server the user is attempting to access. That's encrypted again using the user's key, and sent to the user. The user then transmits that ticket to the server they want to access, along with a request for whatever resource they need.

The server attempts to use its key to decrypt the ticket. If it's able to do so, then several things are known:

- The server is the one the user intended, because if it weren't, it wouldn't have the key needed to decrypt and read the ticket.
- The user's identity is known, because it's included in a ticket *that only the server could read.*
- The user's identify is trusted because the ticket was encrypted not by the user but by the KDC, and in a way that only the KDC and the server could read.

Figure 4.1 shows a functional diagram of how Kerberos works. Keep in mind that this isn't a Microsoft-specific protocol; Microsoft made some extensions to allow for Windows-specific needs—such as the need to include a security token in the tickets—but Windows' Kerberos still works like the standard MIT-developed protocol.

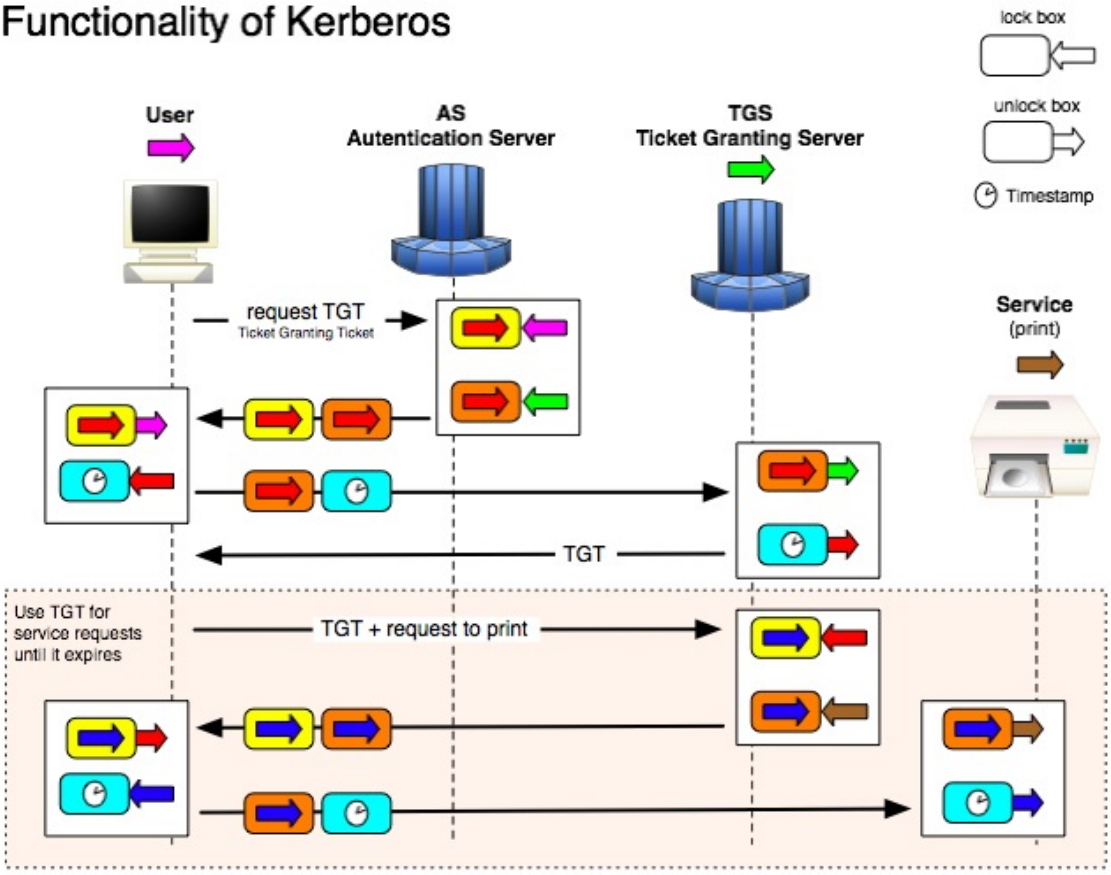

Figure 4.1: Kerberos functional diagram.

The user's computer caches the ticket for 8 hours (by default), enabling it to continue accessing that server over the course of a work day.

> **Note**
>
> If a user's group memberships are changed during the day, that change won't be reflected until the user logs off—destroying their tickets and TGT—and logs back on—forcing the KDC to construct a new TGT.

Microsoft provides a utility called KerbTray.exe, shown in Figure 4.2, which provides a way to view locally-cached tickets.

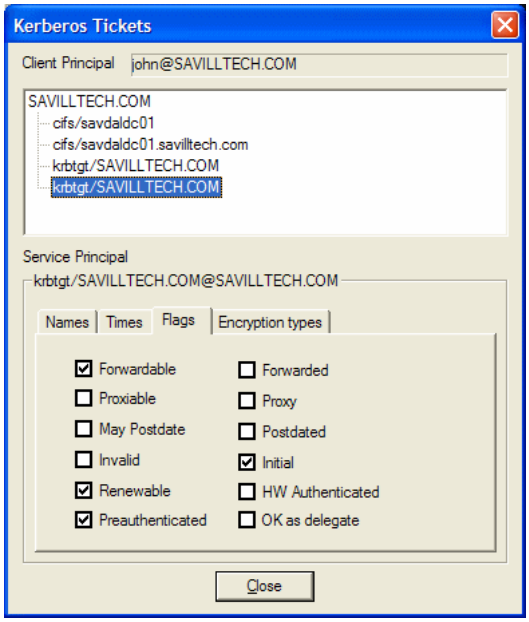

Figure 4.2: The KerbTray utility.

This utility also provides access to several key properties of a ticket, including whether it can be renewed, whether it can be forwarded by a server to *another* server in order to pass along a user's authentication, and so forth.

Kerberos' primary weakness is a dependence on time for the initial TGT-requesting authenticator. In order to prevent someone from capturing an authenticator on the network and then replaying it at a later time, Kerberos requires authenticators to be time-stamped, and will by default reject any authenticator more than a few minutes old. Domain computers synchronize their time with their authenticating domain controller (after authentication), and domain controllers synchronize with the domain's PDC Emulator role-holder. Without this time sync, computers' clocks would tend to drift, taking them outside the few-minutes Kerberos "window" and making authentication impossible.

Authorization: DACLs

As I've already mentioned, AD's main role is authentication. However, for information—such as users and computers, along with configuration objects like sites and services—inside the directory, AD also performs its own authorization and auditing.

Every AD objects is secured with a *discretionary access list.* DACLs follow the same basic structure as Windows' NTFS file permissions. The DACL consists of a list of *access control entries*. Each ACE grants or denies specific permission to a single security principle, which would be a user or a group. Figure 4.3 shows a pretty typical AD permissions dialog.

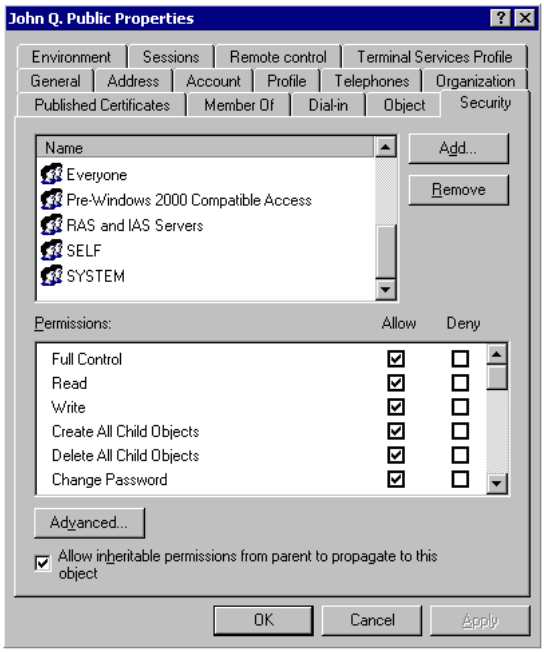

Figure 4.3: AD permissions dialog.

As with NTFS permissions, objects can have directly-applied ACEs in their DACLs, and they can inherit ACEs from containing objects' DACLs. In most directory implementations, for example, user objects have few or no directly-defined ACEs but instead inherit all of their ACEs from a containing organizational unit (OU).

ACEs actually consist of a permissions mask (which defines the permissions the ACE is granting or denying) and a SID. When displaying ACEs in a dialog box, Windows translates those SIDs to user and group names. Doing so requires a quick lookup in the directory, so in a busy network, it's sometimes possible to see the SIDs for a brief moment before they're replaced with the looked-up user or group names.

It's important to understand that, in AD, computers are the same kind of security principle as a user, meaning computers don't have any special permissions. For example, if a Routing and Remote Access Server (RRAS) machine is attempting to authenticate a dial-in user, the server might need to look at properties of the user's AD account to see whether the user has any dial-in time restrictions. Doing so requires that the server have permission to read certain attributes of the user's account, which is why the dialog in Figure 4.2 shows the "RAS and IAS Servers" user group as having permissions to the user's account—without that permission, the server would be unable to examine the user's account to determine whether the dial-in was to be allowed.

Auditing: SACLs

Auditing is defined in *Security Access Control Lists (SACLs)*, which simply define what actions, by which users, will result in a log entry being made in Windows' security log. We'll cover auditing in more detail in the next chapter.

Configuration

AD, like any Windows component, has its own configuration settings, many of which can affect security. For example, consider Figure 4.4, which shows the Group Policy Object (GPO) settings for Kerberos.

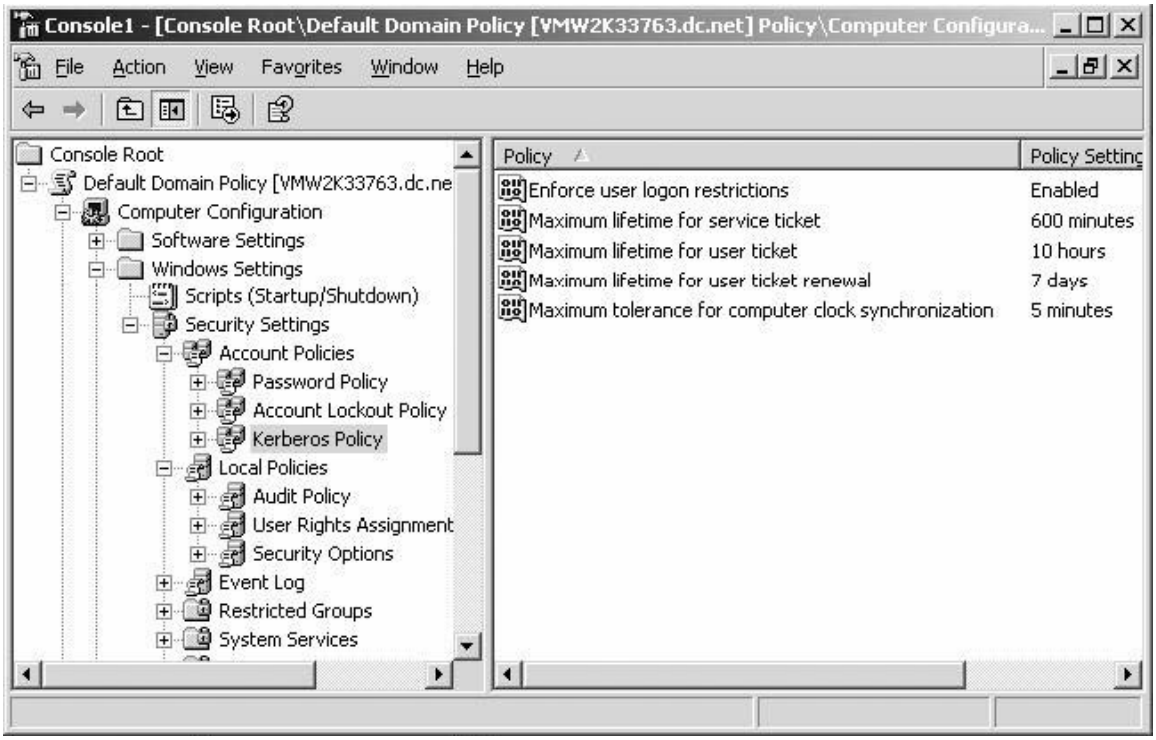

Figure 4.4: Kerberos settings in a GPO.

These settings definitely have a security impact: They control how long a Kerberos ticket is valid, how often it can be renewed, how much time slip is allowed for clock mis-sync, and so forth.

Part of the challenge with AD is that settings like these are scattered all over the place. Some are in the registry and can be modified with a GPO; others live within AD itself, and are accessed by various consoles and command-line tools. Keeping everything straight can be complex; in newer versions of Windows, Microsoft has added a Best Practices Analyzer (BPA), which helps review all of these settings and make recommendations about how to configure them for better security, reliability, performance, and so forth. Figure 4.5 shows an example.

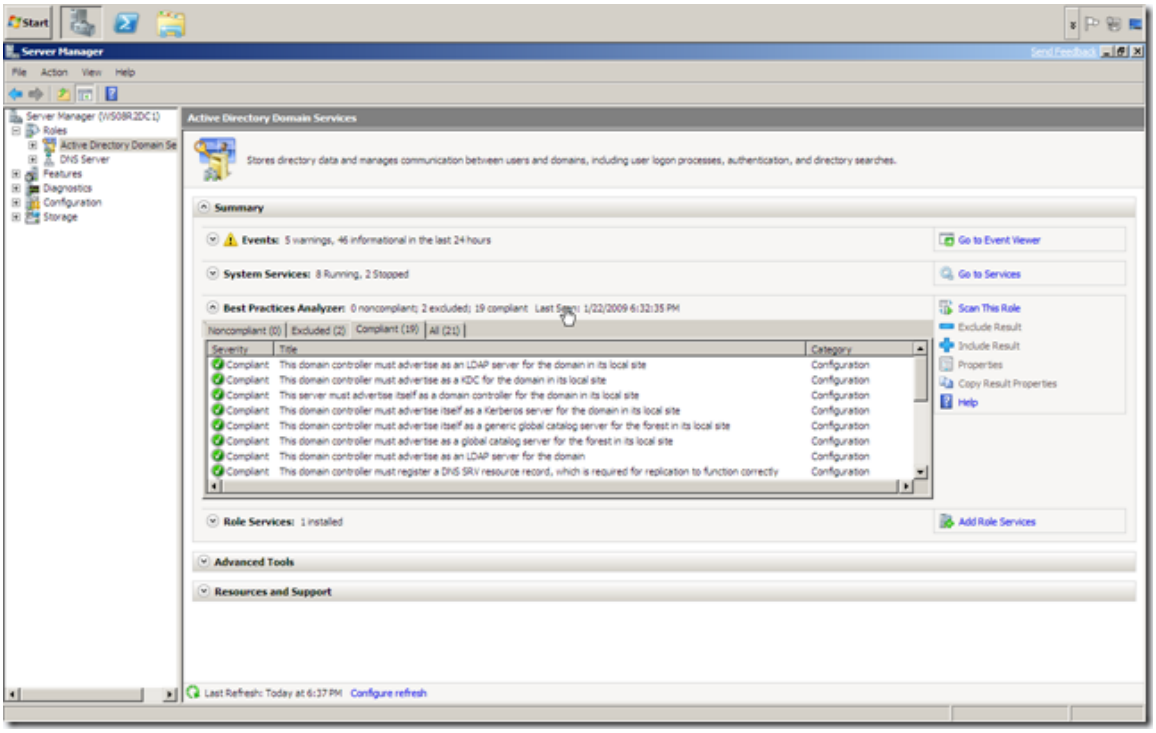

Figure 4.5: A BPA report example.

The "Best Practices" used by this tool are developed by Microsoft, using their own experience with the product, as well as the experiences of major customers. The BPA is new for Windows Server 2008 R2, and the AD model covers a pretty large array of settings. Models are also available for DNS and Certificate Services.

> **Resource**
>
> A complete breakdown of what the BPA scans, and how it works, can be found at http://technet.microsoft.com/en-us/library/dd378893(WS.10).aspx.

Distributed vs. Centralized Permissions Management

AD plays such a central role in authentication that it's easy to forget that the directory really has no role whatsoever in enterprise-wide authorization or auditing. In other words, the directory knows who you are, but it has no clue what you're allowed to do.

This is both a strength and a benefit. With Windows' current architecture, each server maintains its own DACLs on the resources it contains, which might consist of databases, files, mailboxes, or whatever. There's no need to build the robust "central permissions infrastructure" that would be required if servers didn't maintain their own DACLs. Thus, the architecture is better-performing and lower-cost.

Unfortunately, Windows' distribution permissions management evolved when the operating system (OS) was primarily used by small workgroups, not by massive companies with millions of securables. The disadvantage of the distributed permissions management is that certain security questions—such as, "What resources does this user have access to?" are impractical to the point of impossibility. The only way to answer the question would be to manually scan *every single DACL on every single server* to see where that user—or a group he or she was a member of—appears. Doing that on-demand just isn't feasible. And think about it: When a new user starts with a company, someone needs to know what permissions he or she needs. The answer is usually, "Oh, give him the same permissions as so-and-so, who does the same job." The problem is that there's no way to find out what permissions so-and-so has in the first place!

AD's user groups do allow for some degree of centralization *if* an organization's administrators are careful. In other words, if you assign permissions *only* to user groups (which is a practice Microsoft recommends), then you can centrally manage those groups' membership within AD. However, although this practice makes it easier to give a new user the "same permissions as that other guy," it's still impractical to get an inventory of what resources a given group has access to because you still have to scan all of the DACLs. There's also no way of enforcing this practice, and *many* administrators have "put out a fire" by ignoring their organization's groups-only policy and applying an ACE for a single user to a DACL. Over time, these "one-off quick fixes" add up to an impossible-to-manage permissions system.

In fact, *most* Windows-based networks that aren't using some kind of third-party permissions management utility are, in all likelihood, managed very poorly from a permissions perspective. They try to do a good job as much as possible, but the way the distributed system works is simply stacked against them.

There are (as I'll discuss later in this chapter) third-party utilities that can provide that kind of inventory—but they do so by *scanning every single DACL.* They usually do so over several days initially, building a searchable database of permissions. Agents installed on servers can then watch for permissions changes and report those "deltas" to the database, keeping it up to date.

Do-It-Yourself Security Reporting and Changes

Security is one of those things that you're almost constantly looking at for one reason or another. I've already mentioned the BPA, which is a good way to get a basic look at your AD infrastructure's security, performance, and other configuration settings. Without spending any money on third-party tools, you can definitely do some decent reporting.

Permissions

Reporting on permissions is, frankly, hard, due entirely to the way they're stored in Windows. If you want to build your own permissions-reporting tool, you're going to have to scan through a lot of servers. Even answering the question, "What resources can Jill access on this single server?" can be time-consuming because you have to scan through every DACL on the server. Even if most files and folders inherit security from a top-level folder, you can't *assume* that to be the case—you're going to have to *check every file and folder* to make sure.

For that reason, I think building your own permissions-reporting tools is simply impractical. Whatever tools you may have at your disposal—VBScript, Windows PowerShell, and so forth—are going to be too slow to accomplish the task in any reasonable amount of time. Sorry—it's not you, it's Windows.

Directory Objects

Reporting on directory objects—disabled users, old user accounts, locked-out users, and so forth—is easier to do yourself. The AD Users and Computers (ADUC) console provides a Custom Query option that makes this pretty straightforward. As Figure 4.6 shows, you can very easily create a query that shows all users that haven't logged on in, say, the last 90 days—a good starting point for a "stale accounts" report.

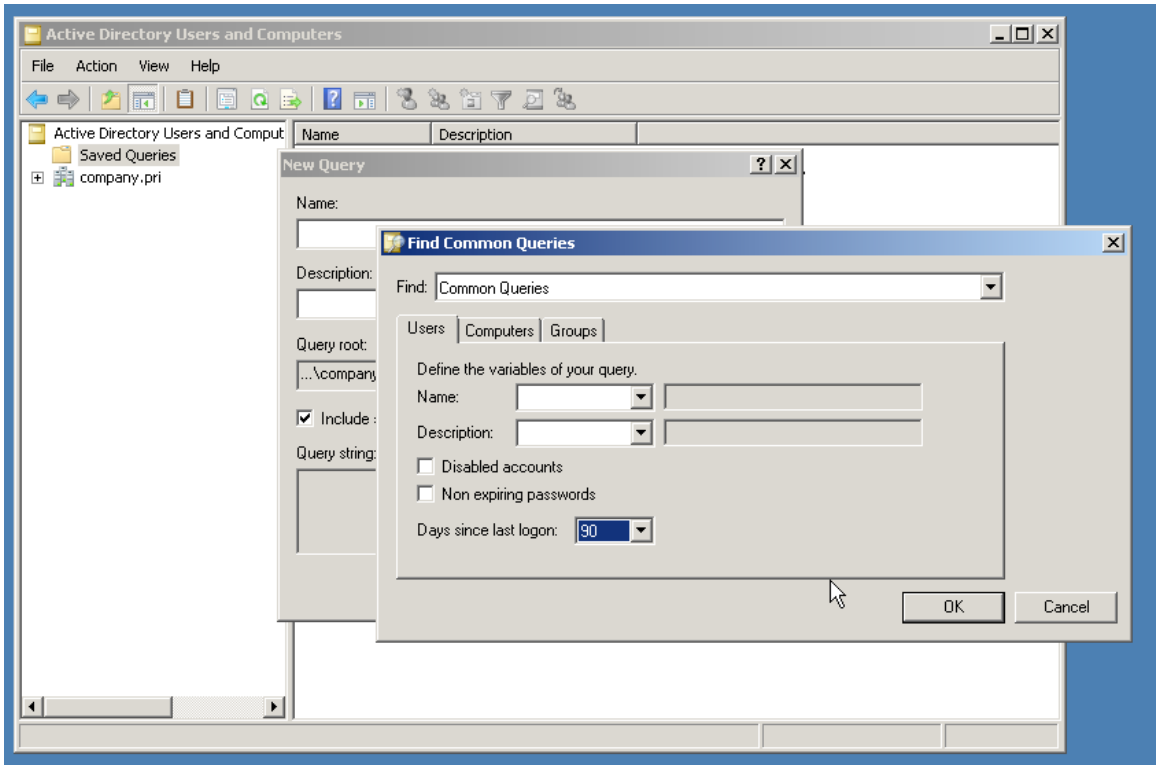

Figure 4.6 Building a custom AD query.

Windows PowerShell can also be used to generate custom reports of a sort. For example, Figure 4.7 shows a PowerShell command that's generating a list of user accounts that have never had their password set. Again, this is a good starting point for other security activities, such as possibly disabling or deleting those accounts.

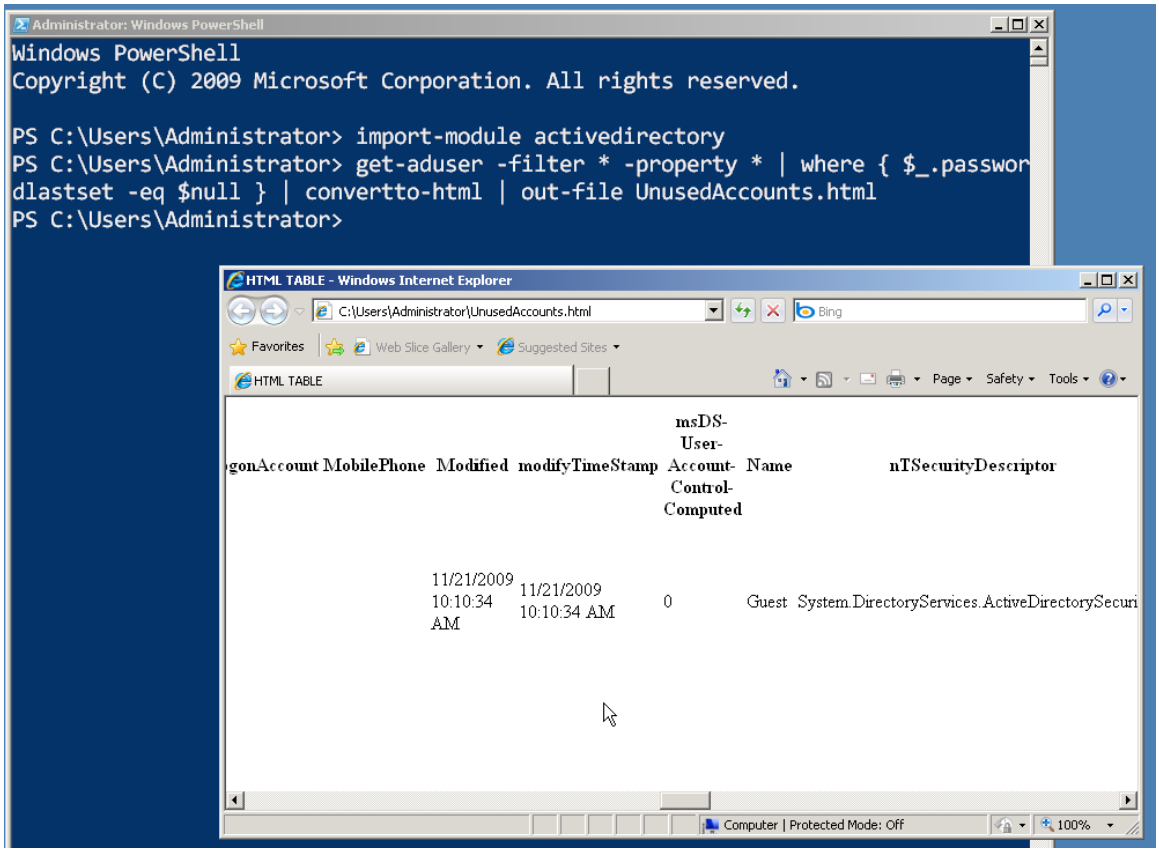

Figure 4.7: Custom reports in PowerShell.

Should You Rethink Your Security Design?

Given the extreme complexity of dealing with permissions on your own, while following best practices, you might want to consider a redesign of your permissions. How you proceed depends a bit upon your goals.

For example, many companies are now moving—or trying to move—to *role-based security.* The idea is that you create a top-level set of roles, which correspond directly to job titles or job responsibilities within your organization. You drop people into those roles, and they pick up the necessary permissions.

In a very small, single-domain environment that has good discipline, you can accomplish this with AD's domain user groups. In larger, multi-domain environments, that becomes a lot harder. Groups are often still used as an under-the-hood means of implementing roles' permissions, but a role will usually be represented by multiple groups because roles span the entire organization, not just a single domain or forest. It's generally considered impossible or at least impractical to implement true role-based permissions in a complex AD environment using only AD's native tools; you generally have to go with a third-party role-based management system that overlays the native AD and Windows security.

Regardless, most companies tend to get really jittery when it comes to redesigning their permissions architecture, mainly because doing so without some kind of third-party tool—which can be expensive—is a daunting task. You have to inventory *everything,* and figure out what resources someone might *need* access to. It's tough. Third-party tools help because they can automate the process at a top-level, taking much of the drudgework and guesswork out of it.

Third-Party Security Capabilities

It's a rare organization that doesn't have some kind of third-party AD tools to supplement its security management. The most common ones fall into the categories of reporting, permissions management, and auditing; we'll save auditing for the next chapter and just briefly focus on the first two.

Reporting

Third-party reporting tools are very common, and can provide a lot of value. Figure 4.8 illustrates one tool, Enterprise Security Reporter, which is designed to report on a number of security-related concerns within AD.

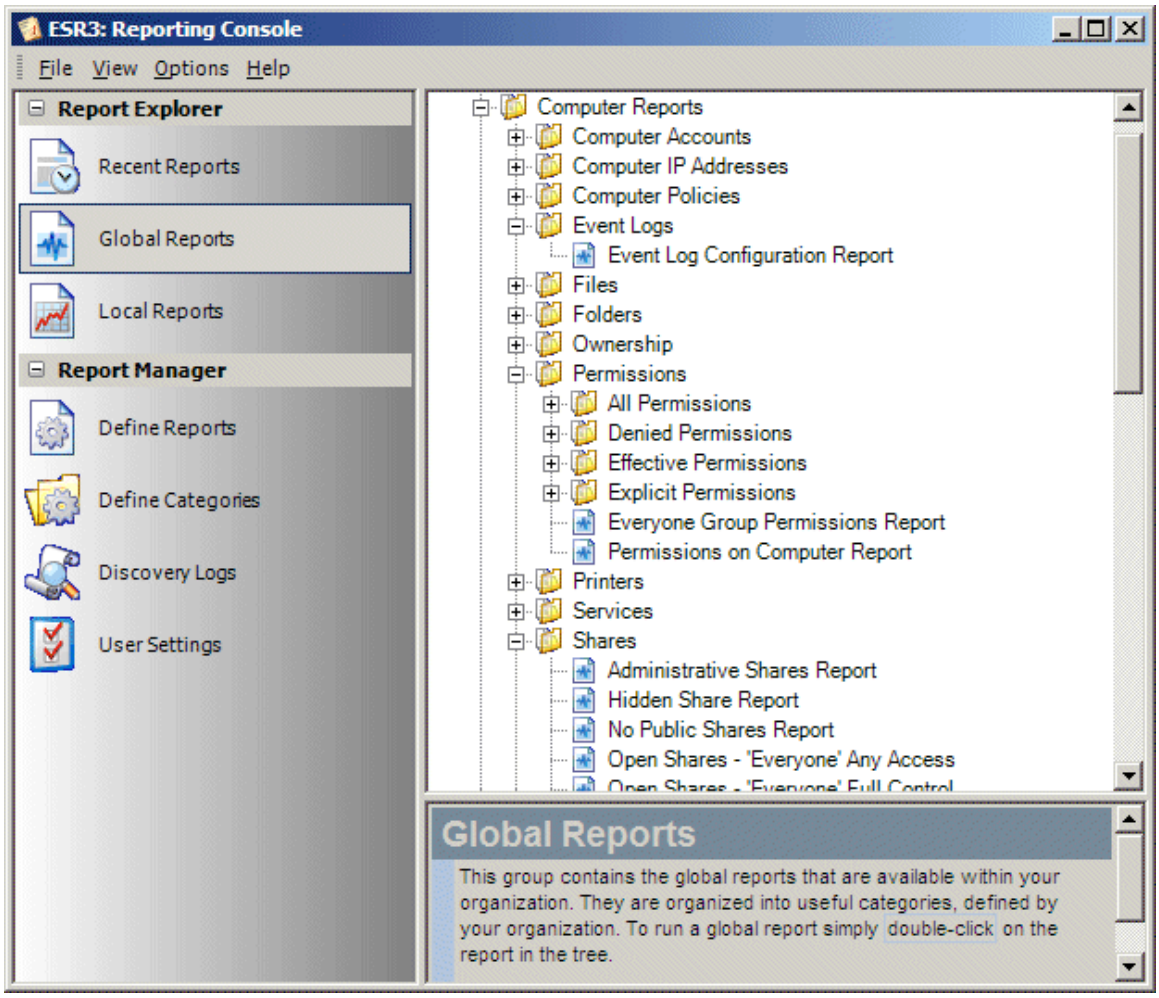

Figure 4.8 A look at the Enterprise Security Reporter.

Figure 4.9 shows another tool, Active Directory Reporter. This tool's focus is broader than security, but it does include a number of security-related reports, as you can see.

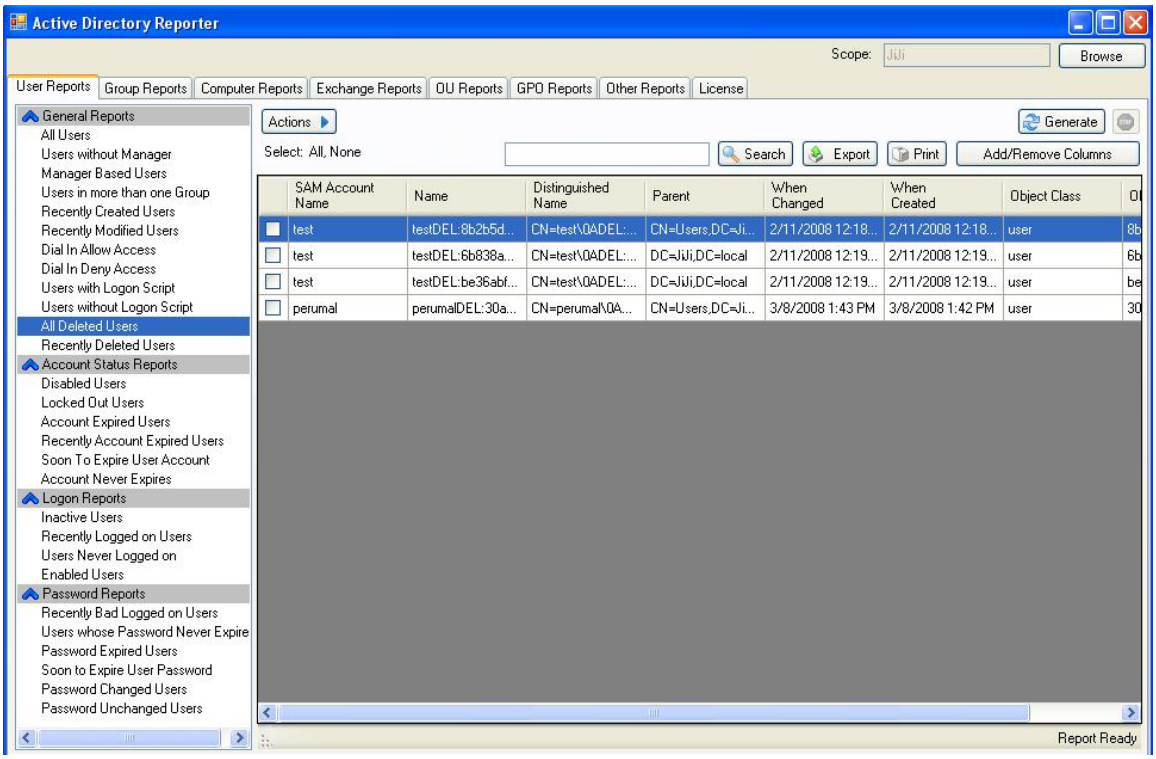

Figure 4.9: The Active Directory Reporter tool.

The idea here is that, rather than spending time (which is money) building your own reporting tools, the right third-party reporting package can give you better-looking and more robust reporting capabilities, making it easier to keep a handle on AD security.

Permissions Management

Third-party permissions management tools typically seek to implement automated role-based permissions for not only AD but also Windows file servers as well as other connected systems like Exchange, SQL Server, SharePoint, and so on. These systems provide a layer on top of the native permissions. They usually start by inventorying existing permissions into a central database. As you make changes to the database's permissions, those changes are pushed out to the relevant resources' native DACLs. Figure 4.10 shows one such tool, called ActiveRoles Server.

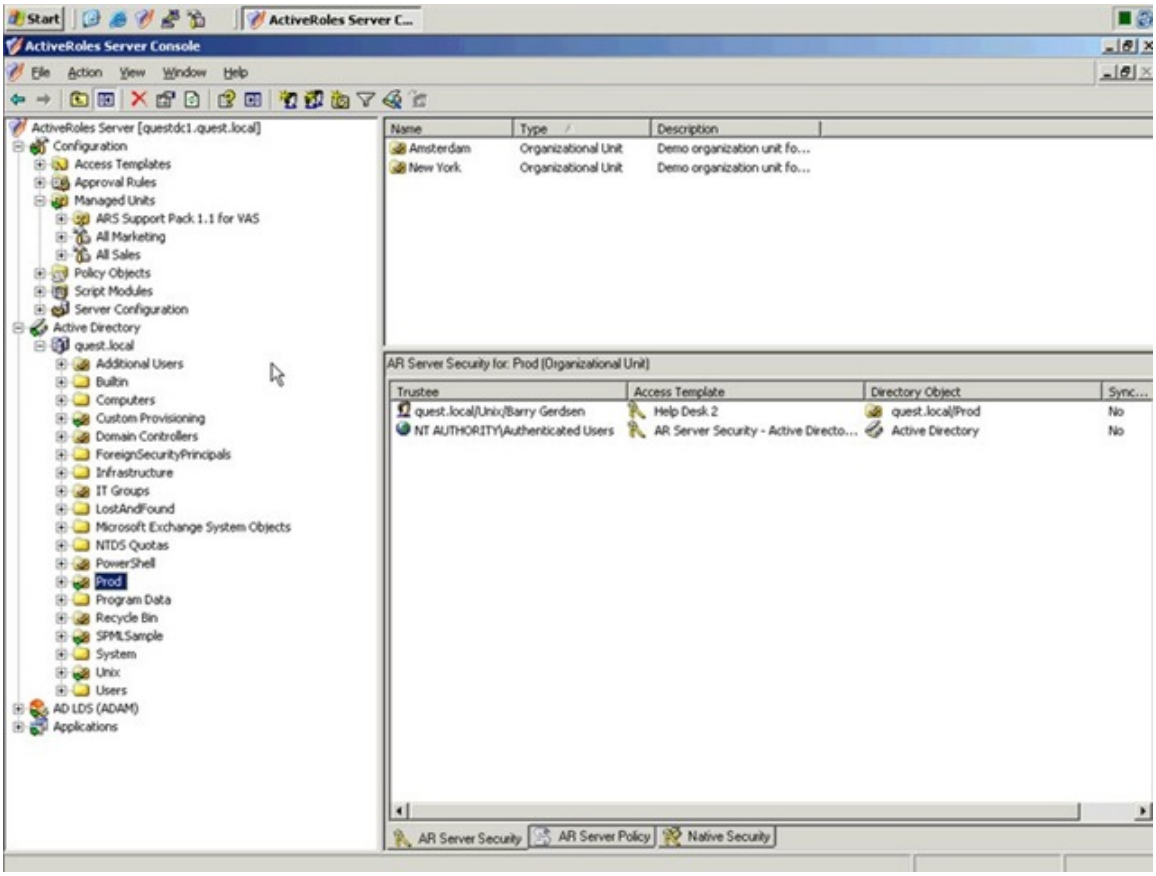

Figure 4.10: An example ActiveRoles Server window.

The idea with most of these tools is that you *stop* managing DACLs directly on resources. Instead, you manage them in the product, enabling it to offer role-based permissions. The product then automates the application of those permissions to the actual resources, giving you centralized control *and* reporting—making it possible to quickly answer questions like, "What resources does Bill have access to?"

DNS Security

The last thing I'll offer in this chapter is an overview of DNS Security, more commonly called *Domain Name System Security Extensions* or simply DNSSEC. DNS obviously plays a vital role in AD's operation, and securing DNS is crucial to maintaining AD's own security and reliability.

The original DNS protocol didn't include any security. Microsoft's implementation of DNS, particularly with the recommended AD-integrated DNS zones, applies a good deal of security by default. Dynamic DNS records are "owned" by their creators and can only be modified by them; other records can have security applied as well. The overall goal of DNSSEC is to prevent forged data from being inserted into the DNS zone database. If someone could do so, they could spoof internal servers and potentially gather sensitive information from unsuspecting users. Although the mutual authentication provided by the Kerberos protocol can help curtail that within a domain environment, Kerberos can't protect non-domain computers, and those could still be spoofed via DNS.

Essentially, DNSSEC works by digitally signing DNS records using digital certificates. Several DNS record types specifically support this activity, including RRSIG, DNSKEY, DS, NSEC, NSEC3, and NSEC3PARAM. When clients make a DNS query, the DNS reply includes not only the traditional A (or AAAA) records, but also RRSIG records that contain a digital signature. The client can then use the DNS server's public key (obtainable in a DNSKEY record) to verify the signature, therefore validating the A or AAAA records.

Relatively few organizations today use DNSSEC, but Windows does support it, and has to a degree since Windows Server 2003. Full support is in Windows Server 2008 R2 and Windows 7. Keep in mind that DNS clients must be DNSSEC-aware in order for the security features to be useful. Non-aware clients can still use a DNSSEC-enabled DNS server, but they will not be able to validate signatures and records.

Why don't more organizations use DNSSEC? Presently, it's not always well-suited in a dynamic DNS environment. For example, creating a signed DNS zone requires you to export an active zone, sign it using a command-line utility (which adds the DNSSEC records to the zone), then load the newly-signed zone as the active database in your DNS server. Dynamic updates are disabled, essentially taking away a key feature that AD relies upon. For that reason, DNSSEC is most often used in external DNS zones, which tend to remain fairly static. That's actually not a bad thing: In a domain environment, DNS is secured by AD and spoofing of domain members is essentially made impossible by Kerberos. In a non-domain environment, where you don't need dynamic DNS, DNSSEC is more practical and meets a need.

Be aware that DNSSEC support is still evolving: The world's DNS root zone doesn't yet support it, nor does the popular .COM top-level domain. Without that support, it's possible to spoof entries in those top-level zones. That support is coming, though. Interim security solutions are available in the meantime, and you can read about them at http://www.windowsitpro.com/article/dns2/DNS-Enhancements-in-Windows-Server-2008-R2/2.aspx. You can read more about Windows' current DNSSEC support at http://technet.microsoft.com/en-us/library/ee649277(WS.10).aspx.

Coming Up Next

Security is one thing: It's how you protect your resources. In the next chapter, we'll look at auditing, the last part of the "AAA" acronym, and a way to keep track of how people are interacting with the security that you've set up.

Download Additional eBooks from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this eBook to be informative, we encourage you to download more of our industry-leading technology eBooks and video guides at Realtime Nexus. Please visit http://nexus.realtimepublishers.com.