

**Windows**  
& .NET MAGAZINE

**TECHNICAL REFERENCE**

# **DNS and Windows 2000**

**Sean Daily  
Gary C. Kessler  
Robert McIntosh  
Mark Minasi  
John Savill  
Tao Zhou**

 **LearnKey**<sup>®</sup>

**Windows & .Net Magazine Technical Reference**

# **A Guide to DNS and Windows 2000**

*By Mark Minasi, Gary C. Kessler, Robert McIntosh,  
Sean Daily, Tao Zhou, and John Savill*



# Windows

& .NET MAGAZINE

Copyright 2004  
Windows & .NET Magazine

All rights reserved. No part of this book may be reproduced in any form by an electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

It is the reader's responsibility to ensure procedures and techniques used from this book are accurate and appropriate for the user's installation. No warranty is implied or expressed.

**ISBN 1-58304-504-X**

## About the Authors

**Mark Minasi** (<http://www.minasi.com/gethelp>) is a contributing editor for *Windows IT Pro*, an MCSE, and the author of *Mastering Windows 2000 Server, 4th Edition* (Sybex). He writes and speaks around the world about Win2K and Windows NT networking.

**Gary C. Kessler** ([kumquat@sover.net](mailto:kumquat@sover.net)) is a consultant and the program coordinator of the networking major at Champlain College in Burlington, Vermont.

**Robert McIntosh** ([rmcintosh@covenantsolutions.com](mailto:rmcintosh@covenantsolutions.com)) is a consultant and trainer who teaches about Microsoft and security technologies and is the founder of Covenant Solutions. He is an MCT, an MCSE, and an ISS-certified instructor.

**Sean Daily** ([sdaily@winnetmag.com](mailto:sdaily@winnetmag.com)) is a senior contributing editor for *Windows IT Pro* and the CEO of Realtimepublishers.com. His most recent books are *The Definitive Guide* series (<http://www.realtimepublishers.com>).

**Tao Zhou** ([tao@winnetmag.com](mailto:tao@winnetmag.com)) is a contributing editor for *Windows IT Pro* and an Internet services engineer for a networking company with headquarters in New Jersey. He is an MCSE, a Master CNE, and holds a master's degree in computer science.

**John Savill** ([john@savilltech.com](mailto:john@savilltech.com)) is a qualified consultant in England and an MCSE. He is the author of *The Windows NT and Windows 2000 Answer Book* (Addison Wesley), and manages the Windows 2000 FAQ Web site (<http://www.windows2000FAQ.com>).

# Table of Contents

<b>Introduction</b> .....	<b>1</b>
<b>Chapter 1: A DNS Primer</b> .....	<b>2</b>
DNS Names and Addresses .....	2
Registering a Name .....	3
Meeting the New Host .....	3
Using the Hierarchy .....	4
Zones vs. Domains .....	4
Locating Mail Servers .....	5
Primary and Secondary DNS Servers .....	5
DNS and Windows 2000 .....	5
Microsoft's DNS Server .....	6
In Your Hands .....	6
<b>Chapter 2: How DNS Works</b> .....	<b>7</b>
Obtaining Names and Addresses .....	7
IP Addressing in the Modern Era .....	7
Sidebar: Subnetting and Variable-Length Subnet Masks .....	8
Registering Domain Names in the Internet's Commercial Era .....	10
DNS Internals .....	10
Tools to Examine DNS and Domain Information .....	12
<b>Chapter 3: Win2K DNS</b> .....	<b>16</b>
The Workings of WINS .....	16
The Differences in DDNS .....	17
<b>Chapter 4: DNS and Active Directory</b> .....	<b>19</b>
Choosing a DNS Server .....	19
Sidebar: DNS Zone Types .....	20
Using Caching-Only Servers .....	22
Centralizing Cache with Forwarders .....	22
Using Split-Brain DNS .....	24
Using Slave Servers to Protect Intranet Servers .....	25
Fitting into an Existing DNS Structure .....	25
Uprooting DNS .....	26

<b>Chapter 5: Navigating Name Resolution</b> .....	<b>28</b>
An Equal Opportunity Annoyer .....	28
Resolution Rudiments .....	28
The Four Node Types .....	28
B-node .....	29
P-node .....	29
M-node .....	29
H-node .....	29
DNS Where You Least Expect It .....	30
Sidebar: Getting Under the Hood with a Network Monitor .....	31
PPTP and Other Problems .....	32
Exceptions to the Rule .....	33
But Wait—There’s More .....	33
Good HOSTS .....	34
Use WINS .....	34
New Order .....	35
Sidebar: Name-Resolution Resources .....	36
Knowledge Is Power .....	38
<b>Chapter 6: Troubleshooting DNS-Related AD Logon Problems</b> .....	<b>40</b>
Finding a DNS Server .....	40
Troubleshooting AD Authentication .....	41
The Origin of the Problem .....	42
A Lack of Communication .....	43
Adding a Second DC .....	44
Building It Right .....	44
<b>Chapter 7: Scavenging Stale DNS Records</b> .....	<b>46</b>
A Time to Live .....	46
And a Time to Scavenge .....	47
Scavenging Periods and Refresh Intervals .....	47
Setting Intervals .....	48
<b>Chapter 8: Integrating UNIX DNS with Windows 2000</b> .....	<b>49</b>
SRV Records in Win2K .....	49
DDNS in Win2K .....	51
Option 1: Migration .....	52
Option 2: Coexistence .....	53
Option 3: Only UNIX DNS .....	55
Ready, Set, Go .....	57

<b>Chapter 9: Maximizing BIND DNS Security</b> .....	<b>.58</b>
DNSSEC Basics .....	.58
The SIG RR .....	.59
The KEY RR .....	.61
The NXT RR .....	.61
Using DNSSEC in BIND .....	.62
Generating Keys .....	.62
Creating a Keyset .....	.62
Signing the Child Zone's Keyset .....	.63
Signing the Zone .....	.63
Using TSIG in BIND .....	.63
Sidebar: A Secure Transaction .....	.64
Looking Forward .....	.66
<b>Chapter 10: DNS FAQs</b> .....	<b>.67</b>
Q. How do I rename a Windows 2000 domain's DNS name? .....	.67
Q. How do I stop my Windows domain controllers (DCs) from dynamically registering DNS names? .....	.68
Q. How do I stop DNS cache pollution? .....	.68
Q. How do I clone Windows DNS zone files? .....	.69
Q. What is the proper use of CNAME records? .....	.70
Q. Why does an Active Directory (AD)-integrated DNS server take longer to start than a typical zone-based DNS server? .....	.71
Q. In a multiple DNS server environment, how do I configure the DNS servers to resolve both local and remote hosts? .....	.71
Q. How do I tell which boot method my DNS service uses? .....	.72
Q. How do I create a chaching-only DNS server? .....	.72

# Introduction

DNS is the most important technology to understand in today's networking environment. At the most basic level, DNS resolves IP addresses to human-readable names. Although computers can easily deal with numeric identifiers such as IP addresses (e.g., 10.255.200.10), people tend to have trouble remembering more than a couple of these identifiers. The TCP/IP DNS protocol lets you associate numeric IP addresses with people-friendly names such as `www.winnetmag.com`. This essential name-resolution service is one of the core technologies that enabled the Internet's rapid growth by letting people easily locate networked resources.

In addition to providing name-resolution support, DNS has evolved into an essential part of the Microsoft Windows networking infrastructure. When Windows NT was the primary Windows OS, DNS was an optional technology—used mainly by organizations with an established UNIX system base. Windows 2000 and Active Directory (AD) brought DNS into the mainstream. Win2K completely incorporated TCP/IP for all aspects of networking, allowing Windows network administrators to drop the old NetBIOS protocol that NT used as a transport and for name resolution. With NetBIOS gone, Win2K moved to TCP/IP's DNS protocol for network name resolution. Microsoft didn't stop halfway in adopting DNS and TCP/IP: DNS is an essential part of AD, and AD completely depends on a functional DNS implementation.

To use DNS effectively, you need to understand its core components. This book starts with a basic foundation for understanding DNS. Chapter 1 offers a DNS primer that introduces DNS components and explains how they interact. Chapter 2 dives a bit deeper to explain exactly how DNS works. Chapter 3 explains the difference between DNS and WINS, as well as some of the DNS extensions that AD uses. Chapter 4 tackles more advanced DNS considerations, such as integrating UNIX and Linux with AD, using DNS cache with forwarders, and using backup DNS servers. Chapter 5 focuses on how the Windows OS uses WINS and DNS to perform name resolution. The book concludes with real-world DNS implementation and troubleshooting information. Chapter 6 covers using Nslookup to troubleshoot AD logon problems that DNS configuration errors can cause. In Chapter 7 you learn about scavenging orphaned DNS records created by dynamic DNS (DDNS). Chapter 8 presents an in-depth guide to integrating UNIX and Windows DNS services. Chapter 9 discusses the issues involved in securing DNS. Finally, the book concludes with some DNS-related FAQs.

## Chapter 1

# A DNS Primer

—by *Mark Minasi*

You might say DNS is an optional feature in Windows NT 4.0. Neither NT nor NT domains rely on DNS. In fact, most organizations running DNS servers don't run those servers on NT—they use UNIX, DNS's traditional OS home. You could theoretically complete an NT 4.0 MCSE certification without strong DNS knowledge.

Windows 2000 offers a different scenario for DNS. If a Win2K workstation needs to find a particular file share on a server, the workstation looks up the server's name in DNS to find the server's network address. Win2K workstations look in DNS to find a domain controller (DC) that they can log on to. (Win2K machines also use DNS to find Global Catalog—GC—servers—another important ingredient to logging on.) Finally, if a workstation must choose from a selection of DCs, DNS helps the workstation find a DC that is nearby rather than one that is a WAN communication away.

DNS is central to Win2K domains' functionality. To use DNS effectively, you need to understand some DNS fundamentals, such as DNS names and addresses, name registration, the DNS hierarchy, primary and secondary DNS servers, and DNS's integration with Active Directory (AD).

## DNS Names and Addresses

Every computing device on the Internet has a unique 32-bit IP address (e.g., 154.23.17.8). When you run Internet-savvy programs, you can refer to these devices by their IP address. For example, you can point your Web browser to <http://68.15.149.117> to access my Web site. However, most of us prefer to point our Web browser to a more human-friendly name such as <http://www.minasi.com>. The ability to use human-friendly names requires a database that can convert [www.minasi.com](http://www.minasi.com) to 68.15.149.117. Converting names to addresses is called name resolution.

Because few machines existed on the Internet in its early days, Internet-attached machines handled name resolution via a simple ASCII table—a HOSTS file—that listed IP addresses and machine names. (TCP/IP software still lets you place a HOSTS file on your system, but you probably won't ever need to use one.) Since 1984, machines on the Internet have chiefly used DNS to resolve names. Imagine needing to maintain on your computer a HOSTS file that not only contains the names of hundreds of millions of computers but also changes daily. With DNS, you don't need to worry about such a scenario.

As the Internet Engineering Task Force (IETF) Request for Comments (RFC) 952 defines, a computer's DNS name consists of several parts separated by periods. For example, [www.minasi.com](http://www.minasi.com) consists of `www`, `minasi`, and `com`. Each part can use no more than 24 characters. The only RFC-approved characters that can go into those name parts are the letters a through z, the numerals 0 through 9, the dash or minus character, and the period that connects the name's

components. In addition, Win2K uses DNS names that use the underscore character—a feature that will affect your choice of DNS servers.

## Registering a Name

To register a domain name, you can go to a central organization called Network Solutions (<http://www.networksolutions.com>). Network Solutions uses many computers to run an Oracle database that keeps track of millions of registered domain names. You can go to the company's Web site and query those computers to find out whether someone has registered a particular domain name (e.g., `acme.com`). Because name registration has become decentralized, you can also register a domain name with groups other than Network Solutions.

DNS's greatest strength is its hierarchically distributed nature. A query of Network Solutions' database will show that someone has registered the name `acme.com`, but the database won't yield any information about that domain. Network Solutions' computers can't tell you whether a particular computer in the `acme.com` domain exists—for example, you can't find out whether a computer named `wile-e-coyote.acme.com` or `meepmeep.acme.com` exists. Even if those computers existed, Network Solutions couldn't give you their IP addresses. Network Solutions neither knows nor cares about what goes on inside `acme.com`, and therein lies the beauty of the hierarchical system. To function, Network Solutions needs only the names of a few `acme.com` contact people, and the names and IP addresses of two computers that will act as DNS servers in that domain.

Suppose the domain name `acme.com` is still available, and you decide to register it with Network Solutions. You'd first give Network Solutions the names and IP addresses of two computers running some kind of DNS server. The computers could be machines inside the `acme.com` network, or you could simply pay your ISP to keep your DNS data on one of the ISP's DNS servers. (One machine can act as a DNS server for many DNS domains.) You decide to run one of these DNS servers locally on a machine called `names.acme.com` (IP address `217.44.93.5`) and pay your ISP to run the other. The ISP puts your domain's DNS information on a server named `ns2.safety-net.net` (`120.10.20.15`). In DNS terms, those two DNS servers are authoritative for the `acme.com` domain.

With the correct software, any computer that runs TCP/IP can be a DNS server. The most popular implementation is UNIX's Berkeley Internet Name Domain (BIND), but I've seen implementations on IBM's mainframes and midrange systems, Digital's (now Compaq's) VAX systems, NT, and OS/2. DNS server implementations for DOS might exist. For the purpose of this chapter, I assume you're running DNS server software on NT.

Now suppose you create machines in the `acme.com` domain with names such as `roadrunner.acme.com`, `www.acme.com`, or `coyote.acme.com`. Remember, you don't need to tell Network Solutions about `roadrunner`, `www`, and `coyote`—the company doesn't care. Rather, you give the information to the two `acme.com` DNS servers (which you told Network Solutions about). Both DNS servers keep a database of information, called a zone file, about `acme.com`.

## Meeting the New Host

The method you use to add a record that describes a new computer—a new host, in DNS terminology—to a DNS server's zone file varies according to the DNS server software that you're running. Most DNS servers use zone files that are ASCII files, so simple file editing would inform a

DNS server of the new machine in town. Other DNS server implementations (e.g., in Win2K and NT 4.0) offer GUI front ends.

Recently designed DNS server software doesn't require you to inform the software about a new host because the software follows a standard called dynamic DNS (DDNS). RFC 2136 describes DDNS in detail. In a DDNS-compliant network, computers can introduce themselves to DNS without requiring an administrator to add the new computers to the DNS zone files.

### Using the Hierarchy

After the acme.com DNS servers have the records for roadrunner, www, and coyote in place, you can see how the DNS hierarchy works. Suppose someone from a domain named products.com points a Web browser to www.acme.com. That browser asks the local domain's DNS server for www.acme.com's IP address. The products.com DNS server doesn't know the answer, but it knows about 13 DNS servers (which Network Solutions and other organizations operate) that know the names and addresses of the DNS servers for each Internet domain. These servers, which are at the root of the DNS tree, are called root servers. The products.com DNS server asks one of the 13 root servers for www.acme.com's IP address. The root server responds that it doesn't know but that the machines that would know are the two authoritative DNS servers for the acme.com domain—the machines with IP addresses 217.44.93.5 and 120.10.20.15. So, the products.com local DNS server opens hailing frequencies to 217.44.93.5 and again asks for www.acme.com's IP address. The machine at 217.44.93.5 responds with www.acme.com's address, and the name resolution is complete.

### Zones vs. Domains

You can further extend DNS's hierarchy. Suppose you have a West Coast office and an East Coast office. Each office wants to run a local DNS server. To accommodate the two offices, you can add a level to acme.com's machine names: Instead of ending with acme.com, each machine's name ends with westcoast.acme.com or eastcoast.acme.com. Each DNS server has a DNS domain subsection (or zone, in DNS jargon). The central acme.com DNS server now keeps the names of only a few hosts. In addition, it keeps track of the fact that two zones now exist, and it stores the names and addresses of the authoritative DNS servers for those zones.

Thus, if the machine named roadrunner is at the West Coast office, the machine gets the name roadrunner.westcoast.acme.com. The offices have separate DNS servers, so a name resolution query from outside the organization would require another round of DNS server queries. A products.com DNS server that wants to retrieve roadrunner.westcoast.acme.com's IP address would first query Network Solutions for the addresses of the acme.com DNS servers. The main acme.com DNS server, unable to provide the IP address, would direct the products.com DNS server to the main acme.com DNS server. The main acme.com DNS server, recognizing that the query is about a West Coast machine, refers the query to the specific DNS server for the West Coast. Finally, that West Coast DNS server gives the products.com DNS server the IP address of westcoast.acme.com's DNS server.

The name structure is completely hierarchical. When you separated acme.com to two coasts, you didn't need to inform Network Solutions. Systems that need an address from one coast simply discover where to find addresses as they thread their way through the DNS hierarchy.

Although I describe a hierarchy of names with the root servers at the top, you needn't use that hierarchy. If your network isn't connected to the outside world, you can define your own top-level or root servers.

## Locating Mail Servers

Another useful DNS feature is the ability to locate a domain's email servers. If I have a user account called mark in the acme.com domain, I'd like people to be able to send mail to mark@acme.com. But email needs to go from email server to email server. Suppose I have a friend named Tom at domain apex.com, and Tom wants to send me an email message. His email server must find acme.com's email server. How does this server-to-server rendezvous take place?

To answer that question, consider how you find servers in another popular Internet application—the Web. If you want to examine acme.com's Web page, how do you find the Web server's address? You'd guess that its name is www.acme.com (and you'd probably be right), but a more formal way to find the name of a Web server in a given domain would be nice. No such capability exists, but you can look up the name of an email server in a specific domain. Zone files can contain a mail exchanger (MX) record that names the zone's email server.

## Primary and Secondary DNS Servers

You've probably guessed the reason why you need two DNS servers—reliability. If one server goes down, the other can perform acme.com's name-resolution functions. But how do you keep the servers synchronized? If you add a machine named yosemitesam.acme.com to the domain, how do you ensure that both the names.acme.com DNS server and the ns2.safety-net.net DNS server receive that information? Simply designate one server to be a primary DNS server and the other to be a secondary DNS server.

I do all my acme.com zone file editing on the primary DNS server. Then, the primary server sends the most recent zone file information to the secondary server. When a machine queries the secondary server about acme.com, the secondary server answers the query from its copy of the zone file. The secondary server's zone file typically has a Time to Live (TTL) period. If the primary DNS server doesn't update the secondary server's zone file within this expiration period (i.e., a certain number of hours), the secondary server assumes that all the zone information is stale and disregards it. The TTL period is typically 24 hours or more, so if the primary DNS server is down for only a few hours, you won't experience a problem.

Which DNS server is the primary server, and which is the secondary server? Network Solutions doesn't know or care; you won't find that information in the company's database. You determine which DNS server is the primary name server in the Start of Authority (SOA) record. The SOA record also tells secondary DNS servers how long to keep old zone file information, specifies the TTL period, and reports the email address of the domain's technical contact.

You can have as many secondary DNS servers as you want. This scenario might sound familiar. In many respects, primary and secondary DNS servers are similar to PDCs and BDCs in NT 4.0 and NT 3.x.

## DNS and Windows 2000

With MX records, standard DNS lets you easily find the email servers for a particular domain. RFC 2052 lets you look up other kinds of servers via a new kind of DNS record—an SRV record. An

SRV record lets you ask a DNS server whether it knows of any machines that act as servers of a specific type. Win2K-aware machines look in the SRV records to find DCs.

Win2K-aware machines exploit another DNS feature: DDNS. When a Win2K client machine starts, it asks its local DNS server to add the client name to the zone database. Although pre-Win2K systems don't know to ask the DNS server to add them to the zone database, Win2K's DHCP server accomplishes the task for them. So, even legacy machines get added to the local zone file. Win2K blurs the line between NT domain and DNS domain. A DNS zone file under Win2K functions similarly to a WINS database under NT 4.0 and earlier.

Win2K uses DDNS in another way. When you create or modify an AD-based domain, the DC automatically reflects that domain's structure in the domain's zone files. And the DC uses DDNS to enter those records into the DNS database.

### Microsoft's DNS Server

To benefit fully from the integration of DNS and AD, you don't need to use the DNS server that ships with Win2K. However, you can't continue using a 1992 copy of BIND.

To work with Win2K, DNS servers must satisfy a couple of prerequisites. First, the DNS servers must support RFC 2052 (the SRV record type) and RFC 2136 (DDNS). Many current DNS server products (including the current version of BIND) offer such support. Second, the DNS server must accept host names that include the underscore character. Many of the records that AD automatically creates include underscores. Because DNS doesn't accept underscores by default, many recent DNS implementations won't accept DDNS registrations on names that include underscores. Alternatively, you can divide your existing DNS domain into two or more zones, place the Win2K and NT machines into a new zone, make a Win2K system the authoritative DNS server for only that zone, and leave all your other machines in the old zone.

### In Your Hands

To fully understand DNS, you need to know how the international DNS hierarchy functions, how new RFCs extend DNS's power, and how you can fit your organization into the DNS hierarchy. All these pieces fit together to make DNS the perfect choice for a Win2K naming infrastructure.

## Chapter 2

# How DNS Works

—by Gary C. Kessler

As a systems administrator, you might be responsible for connecting your company to the Internet, possibly for the first time. But before you can do so, you need to understand the basics of Internet addressing, find out how to get a Web address, and learn how to register your domain name. In Chapter 1, Mark Minasi introduced the fundamentals of DNS. I will help you find resources for obtaining IP numbers and registering your domain name and provide practical information about DNS and its associated tools.

## Obtaining Names and Addresses

The Internet Assigned Numbers Authority (IANA—<http://www.iana.org>) was historically the organization that doled out names and addresses. However, in 1993, the National Science Foundation (NSF) awarded to Network Solutions (<http://www.networksolutions.com>) a 5-year contract that authorized Network Solutions to operate the InterNIC (<http://www.internic.net>) name-registration service. At that time, InterNIC was the place that most people went to for a network identifier (NET\_ID) or a domain name in the .com, .org, or .net namespace. When InterNIC assigned your IP address, you owned the address and could keep it even if you changed ISPs. Today, the process of obtaining addresses and of registering domain names has changed.

## *IP Addressing in the Modern Era*

InterNIC no longer assigns IP addresses and hasn't done so since about 1997. In the Western Hemisphere and some parts of the African continent, the American Registry for Internet Numbers (ARIN—<http://www.arin.net>) is now the IP number authority. The size of Internet routing tables was becoming unmanageable, so several years ago ARIN organized unused address space into Classless Inter-Domain Routing (CIDR) address blocks. ARIN assigns the CIDR address blocks, in turn, to ISPs or other regional number authorities, such as Réseaux IP Européens (RIPE—<http://www.ripe.net>) in Europe and the Asia Pacific Network Information Centre (APNIC—<http://www.apnic.net>). If you're in the United States, you need to coordinate with your ISP to obtain an IP NET\_ID address from ARIN.

More than 50 percent of all possible Class C addresses are still available, but because the address supply is rapidly diminishing, getting a full Class C address is difficult. Instead, ISPs and ARIN distribute small address blocks and assign NET\_IDs with as few as eight host addresses. IP number authorities assign the address blocks by using variable-length subnet masks (VLSMs), which provide a method of implementing classless addressing. A full description of VLSMs is beyond the scope of this chapter, but the sidebar “Subnetting and Variable-Length Subnet Masks” describes the motivation for using VLSMs and gives a VLSM example.

## Subnetting and Variable-Length Subnet Masks

IP addresses are 32 bits long. The length of the network identifier (NET\_ID) part of the address determines an IP address classification. Class A, B, and C addresses have 8-, 16-, and 24-bit NET\_IDS, respectively. For routing purposes, IP devices such as hosts or routers must separate the NET\_ID from the host identifier (HOST\_ID) portion of the address. Subnet masks perform this separation.

Subnet masks are usually a string of 1s that denote the number of bits of the address that forms the NET\_ID. A Class A address, for example, has an 8-bit subnet mask, which is written 255.0.0.0. Class B and Class C addresses use the subnet masks 255.255.0.0 and 255.255.255.0, respectively. Subnet masks can also segment the HOST\_ID. Some sites use the HOST\_ID to create a subnetwork identifier and smaller host identifier.

In the past several years, IP address depletion has become a problem. Address depletion occurs not only because so many new networks exist but also because IP address authorities are inefficiently assigning classful IP addresses. A site requiring 500 addresses, for example, might receive an entire Class B block of more than 65,000 addresses.

To solve this problem, Internet address administrators have adopted variable-length subnet masks. VLSMs allow NET\_IDS of lengths between 8 and 30 bits. (Subnet masks don't allow lengths of 9, 15, 17, 23, or 25 bits because they look like single-bit subnets.) VLSMs build flexibility into the IP addressing system, and ISPs can efficiently assign address space.

For example, suppose the U.S. Cable Networks, a consortium of US cable TV companies, receives the Class A address 24.0.0.0. Because 24.0.0.0 is a Class A address, you can denote it as 24.0.0.0/8 (the /8 is a shorthand notation for the 8-bit subnet mask 255.0.0.0). Several cable TV company ISPs share the 24-block address. Adelphia Cable Communications (whose ISP service was formerly Hyperion) receives the NET\_ID block 24.48.0.0. This block is equivalent to a Class B address, denoted as 24.48.0.0/16. Adelphia's ISP customers, in turn, receive IP addresses from the 24.48.0.0 space and, using VLSM, usually get something smaller than a Class C address space.

For example, Pizzagalli Construction in Burlington, Vermont, receives a block of 64 addresses with the NET\_ID 24.48.165.0/26 (i.e., subnet mask 255.255.255.192). The host addresses (the final digit set in the NET\_ID) range from 0 to 63. Pizzagalli can't use 0 or 63 because addresses that contain all 0s or all 1s are invalid host addresses. In a final example, Pension Works in Colchester, Vermont, receives a block of 32 addresses. The company's NET\_ID is 24.48.165.64/27 (i.e., subnet mask 255.255.255.224), and the company's host address range is 64 to 95. Usable host addresses are 65 to 94.

Continued on page 11

**Subnetting and Variable-Length Subnet Masks** *continued*

Because sub-Class C addresses are common today, you need to understand how subnet masking works. Table A shows the subnet masks that you might use with a Class C address. Assigning sub-Class C addresses preserves address space, but VLSMs are also beneficial because they enable Classless Inter-Domain Routing (CIDR). The American Registry for Internet Numbers (ARIN) introduced CIDR several years ago to reduce the size of Internet routing tables. Customers used to receive assigned IP addresses sequentially. Now ISPs assign IP addresses from a block of addresses. Now our example company, Adelphia Cable Communications, can get all its packets from the 24.48.0.0/16 network. The hundreds or thousands of networks can reference one line in a routing table. After the packets get to Adelphia, the company's routers have to deliver the packets to the correct destination network, but that detail is transparent to the rest of the Internet.

**TABLE A:**  
*Subnet Masks You Can Use with a Class C Address*

Number of Bits in Mask	Subnet Mask	Number of Host Addresses	Notes
24	255.255.255.0256	(254 usable)	Standard full Class C
25	255.255.255.128	-	Single-bit subnet masks (never used)
26	255.255.255.192	64 (62 usable)	Common
27	255.255.255.224	32 (30 usable)	Common
28	255.255.255.240	16 (14 usable)	Becoming more common
29	255.255.255.248	8 (6 usable)	Becoming common
30	255.255.255.252	4 (2 usable)	Typically used on point-to-point link between ISP and customer
31	255.255.255.254	-	Invalid; results in single-bit HOST_ID
32	255.255.255.255	-	Invalid; no room for HOST_ID

How can a company with several hundred systems operate with a mere handful of IP addresses? Companies might have many users, but generally they have only a few servers that require publicly visible IP addresses. Increasingly, organizations use private IP addressing internally and assign a public IP address to public servers. Companies might use the Internet Engineering Task Force (IETF) Request for Comments (RFC) 1918 private addresses or Network Address Translation (NAT) to statically map the server's public address to an internal private address. When a client system communicates with the Internet, NAT dynamically and temporarily assigns the client a public address. If the number of clients that need addresses exceeds the number of available public addresses, Port Address Translation (PAT—aka NAT overload) provides the addresses. NAT and PAT address management occur transparently at the router or NAT server.

## ***Registering Domain Names in the Internet's Commercial Era***

The NSF's contract with Network Solutions expired in April 1998. Unfortunately, no one planned how to handle domain name requests after that date, so NSF has extended Network Solutions' contract several times. In 1998, the Internet Corporation for Assigned Names and Numbers (ICANN—<http://www.icann.org>) began creating a fair and efficient domain-name-registration system. Many factors influenced the formation of ICANN, and you can find a good description of its history and evolution at the National Telecommunications and Information Administration (NTIA) Web site (<http://www.ntia.doc.gov/ntiahome/domainname>). Although Network Solutions remains the sole administrator of names in the .com, .org, and .net namespace, many companies, such as America Online (<http://www.aol.com>), ItsYourDomain.Com (<http://www.itsyourdomain.com>), and Register.com (<http://www.register.com>), can register names in that space. You can find a list of all accredited name registrars at the ICANN Web site.

## **DNS Internals**

DNS is a distributed database that contains host, mail server, name server, and other domain information. You must maintain a primary name server and at least one secondary name server for every Internet domain. When a client system on the Internet needs to find a server's IP address, the client sends a DNS query to its local name server. If the local name server doesn't have the necessary target server address information, the local name server sends a query to one of 13 well-known root name servers on the Internet. The query then proceeds to one of the target domain's name servers for final resolution. Many ISPs provide primary and secondary DNS service; others provide only secondary DNS service and require the customer to host the primary name server.

Examining DNS file structure will help you understand the name-lookup process. DNS information resides in simple text files called zone files, which contain information called Resource Records (RRs). The most common RRs are

- Start of Authority (SOA)—denotes the primary name server for a domain and a few additional administrative items
- Address (A)—supplies a host name's IP address
- Canonical Name (CNAME)—provides alias host names so that you can associate more than one host name with an IP address
- Pointer (PTR)—associates a host name with an IP address and performs reverse name lookups
- Mail Exchanger (MX)—defines a domain's mail systems
- Name Server (NS)—defines a domain's name servers

Listing 1 shows an example zone file mapping host names to IP addresses in the fictitious example.com domain. In this example, the domain hosts Web, FTP, email, and name servers, and the ISP (ispexample.net) hosts the secondary DNS and backup mail server. The IP address in this example is actually a private IP address. So what do all these records mean? The SOA record includes the name of the primary DNS server for this domain and the email address associated with this domain's naming administrator; note that the record lists the DNS administrator's email with a period (.) instead of the at sign (@) because @ has special meaning in DNS files.

**LISTING 1:***Example Zone File for the Example.com Domain*

```

example.com.      IN SOA  dns.example.com.  dnsowner.example.com. (
19991005 ;serial # (date format)
10800 ;refresh (3 hours)
3600 ;  retry (1 hour)
604800 ;expire (1 week)
86400) ;  TTL (1 day)
example.com.      IN NS   dns.example.com.
IN NS ns1.ispexample.net.
example.com.      IN MX   20      mail.example.com.
IN MX   40      mail.ispexample.com.
dns.example.com.  IN A    192.168.210.2
mail.example.com. IN A    192.168.210.4
www.example.com.  IN A    192.168.210.5
ftp.example.com.  IN CNAME www.example.com.

```

This RR also contains five other parameters. First, a serial number identifies the version of this information and tells a secondary server that new information exists to download. Second, a refresh value tells the secondary name servers how often to check for updated information. Third, a retry value tells the secondary servers how often to reattempt connections to the primary server. Fourth, an expire value tells the secondary servers when the information in databases is old and unreliable. And fifth, a Time to Live (TTL) value tells a requester how long you can safely cache the information.

The NS records contain the names of the name servers for this domain. The first server listed is the primary name server because it's the server that the SOA record names. The MX records contain the names of the email servers for this domain. The number in front of the address is the preference value and is most useful when the domain has two or more email servers.

When a remote user sends mail to `user@example.com`, the remote mail system looks up the MX record for the `example.com` domain. The remote mailer then attempts to establish an SMTP connection with the mail server that has the lowest preference value. Thus, an organization can specify multiple mail servers with the same preference level for load balancing, or specify servers with different preference levels to provide a backup. The A records contain the IP addresses to associate with each of the listed host names in the `example.com` domain. The CNAME record contains alias host names. In this example, the FTP and Web services are on the same server but have two different names that map to the same IP address. Using two separate names ensures that if the FTP service moves to another system, external users never need to know about the move.

Each domain contains another important zone file: the reverse lookup file. This file maps an IP address to a host name. The zone file for the fictitious `192.168.210.0` address space (assuming that this entire Class C address has one owner) might look like the file in Listing 2. The only new RRs in this listing are the PTR records, which associate an IP address with a host name.

**LISTING 2:***Example Reverse Lookup File for the Example.com Domain*

```

210.168.192.in-addr.arpa. IN SOA dns.example.com. dnsowner.example.com.
(19990930 ;serial # (date format)
10800 ;refresh (3 hours)
3600 ; retry (1 hour)
604800 ;expire (1 week)
86400) ;minimum TTL (1 day)
210.168.192.in-addr.arpa.                IN NS dns.example.com.
IN NS ns1.ispexample.net.
2.210.168.192.in-addr.arpa.              IN PTR    dns.example.com.
4.210.168.192.in-addr.arpa.              IN PTR    mail.example.com.
5.210.168.192.in-addr.arpa.              IN PTR    www.example.com.

```

You'll find DNS server software under a variety of names, depending on the OS you use. Windows NT simply refers to DNS, but UNIX calls the software named (i.e., name daemon) or Berkeley Internet Name Domain; BIND is the most common name. Although each DNS software package is slightly different from others, it's useful to understand RRs and file formats so that you know how the packages lay out domain name information.

All DNS software uses the same terminology to refer to DNS information, whether the software uses regular DNS text-file format or a proprietary-file format. Furthermore, although individual sites might not use UNIX-based DNS software, the major ISPs do, so you need to use the correct terminology when discussing DNS with your ISP.

Last but not least, UDP datagrams on port 53 carry DNS queries. DNS zone transfers between primary and secondary name servers use TCP on port 53. If you run the primary DNS server on your network and connect to an outside secondary DNS server, configure your firewall so that zone transfers can occur only between the designated name servers.

## Tools to Examine DNS and Domain Information

The most basic tool you can use to search a domain name or IP number database is Network Solutions' Whois. Although you'll find Whois on all UNIX systems and some Windows systems, accessing the database is easiest on the Internet. You can find Network Solutions' Web interface to Whois at <http://www.networksolutions.com/cgi-bin/whois/whois>. Figure 1 shows a Whois query for the win2000mag.com domain. The response shows contact information for the domain, when the record was last updated, and the associated name servers (in preference order). You can use Whois to look up information based on domain names and contact names. Whois can also tell you whether a particular domain name is available.

**Figure 1:**  
*Viewing the Web interface to Network Solutions' Whois*



Nslookup is a handy TCP/IP utility for examining the DNS database. This utility is a standard part of NT (and UNIX) systems, and versions of Nslookup for Windows 9x also exist. You can use the Nslookup utility as a test aid to examine DNS.

Figure 2 shows an example Nslookup session. As callout A in Figure 2 shows, the user invokes the program by typing the command `nslookup`. The program responds by listing the name and address of the user's default name server. The first command, `help`, which callout B in Figure 2 shows, lists all Nslookup commands and functions. Callout C in Figure 2 shows that the user next enters host name `www.win2000mag.com`. The program responds by listing the host's IP address (`204.56.55.202`). The set type=`MX` command, which callout D in Figure 2 shows, tells the program to display MX information. The following command, `win2000mag.com`, asks for information about the `win2000mag.com` domain. The program responds with the names and addresses of the domain's three mail servers (and two name servers). Callout E in Figure 2 shows the set type=`SOA` and `win2000mag.com` commands, which tell the program to display SOA information about the `win2000mag.com` domain. Finally, the program responds with the SOA parameter information, as well as the names and addresses of the domain's name servers.

**Figure 2:**  
*Screen Output of Example Nslookup Terminal Session*

```

A C:\>nslookup
Default Server: clover.sover.net
Address: 209.198.87.40

B > help
Commands: (identifiers are shown in uppercase, [] means optional)
NAME - print info about the host/domain NAME using default server
NAME1 NAME2 - as above, but use NAME2 as server
help or ? - print info on common commands
set OPTION - set an option
  all - print options, current server and host
  [no]debug - print debugging information
  [no]d2 - print exhaustive debugging information
  [no]defname - append domain name to each query
  [no]recurse - ask for recursive answer to query
  [no]search - use domain search list
  [no]lvc - always use a virtual circuit
  domain=NAME - set default domain name to NAME
  srchlist=N1[/N2/.../N6] - set domain to N1 and search list to N1,N2, etc.
  root=NAME - set root server to NAME
  retry=X - set number of retries to X
  timeout=X - set initial time-out interval to X seconds
  querytype=X - set query type, e.g., A,ANY,CNAME,MX,NS,PTR,SOA
  type=X - synonym for querytype
  class=X - set query class to one of IN (Internet), CHAOS, HESIOD or ANY
server NAME - set default server to NAME, using current default server
lserver NAME - set default server to NAME, using initial server
finger [USER] - finger the optional NAME at the current default host
root - set current default server to the root
ls [opt] DOMAIN [> FILE] - list addresses in DOMAIN (optional: output to FILE)
  -a - list canonical names and aliases
  -d - list all records
  -t TYPE - list records of the given type (e.g. A,CNAME,MX,NS,PTR etc.)
view FILE - sort an 'ls' output file and view it with pg
exit - exit the program

C > www.win2000mag.com
Server: clover.sover.net
Address: 209.198.87.40

Name: www.win2000mag.com
Address: 204.56.55.202

D > set type=MX
> win2000mag.com
Server: clover.sover.net
Address: 209.198.87.40

win2000mag.com MX preference = 30, mail exchanger = mail2.rockymtn.net
win2000mag.com MX preference = 10, mail exchanger = mail.duke.com
win2000mag.com MX preference = 20, mail exchanger = mail.rockymtn.net

win2000mag.com nameserver = ns1.duke.com
win2000mag.com nameserver = ns2.duke.com
mail2.rockymtn.net internet address = 166.93.8.2
mail.duke.com internet address = 204.56.55.3
mail.rockymtn.net internet address = 166.93.8.2
ns1.duke.com internet address = 204.56.55.1
ns2.duke.com internet address = 204.56.55.2

E > set type=SOA
> win2000mag.com
Server: clover.sover.net
Address: 209.198.87.40

win2000mag.com
primary name server = ns1.duke.com

```

Continued on page 16

Figure 2 continued

```
responsible mail addr = dnsadmin.duke.com
serial = 10
refresh = 3600 (1 hour)
retry = 600 (10 mins)
expire = 86400 (1 day)
default TTL = 3600 (1 hour)

win2000mag.com nameserver = ns1.duke.com
win2000mag.com nameserver = ns2.duke.com
ns1.duke.com internet address = 204.56.55.1
ns2.duke.com internet address = 204.56.55.2
> exit

C:\>
```

The NS host information that Nslookup shows (ns1.duke.com and ns2.duke.com) doesn't match the host information that the Whois lookup shows (ns1.rockymtn.net and ns2.rockymtn.net). This result is unusual but merely signals that the Whois database isn't synchronized with the name server information advertised on the Internet. Given this discrepancy, the display from Nslookup is more definitive than the display from Whois.

In the past, Internet connections, IP addresses, and domain names were the responsibility of UNIX systems administrators. However, because NT represents a growing percentage of servers on the Internet, these details are important to systems administrators who have traditionally concentrated on the LAN. DNS is one of the most important aspects of your site's Internet connection. If you don't set it up correctly, your public hosts might be unreachable and your users might not be able to reach hosts on the Internet. If you want to know the nitty-gritty behind DNS, the industry-standard text is Paul Albitz and Cricket Liu's *DNS and BIND*, 4th edition (O'Reilly & Associates, 2001). For NT-specific information, see Matt Larson and Cricket Liu's *DNS on Windows 2000*, 2nd edition (O'Reilly & Associates, 2001).

## Chapter 3

# Windows 2000 DNS

—by *Mark Minasi*

In some ways, Windows 2000 DNS seems to be merely a replacement for WINS under Windows NT 3.51 and later. Because NT 4.0 doesn't depend much on DNS, most of us don't think much about our DNS infrastructure. In contrast, WINS under NT requires quite a bit of thought, in particular about how many WINS servers you need. Having many WINS servers makes for speedy name resolution, which is good. But having many WINS servers also leads to replication and database-corruption problems, which is decidedly bad. Consequently, WINS architecture is something of a balancing act.

In comparison, DNS is generally a more trouble-free naming service than WINS. In this chapter, I explain why.

## The Workings of WINS

WINS and DNS are both name-resolution systems, which means that they keep track of the name and IP address of every system on your network. That tracking is a great convenience: Because WINS and DNS know both the name and IP address of all your computers, you can assign your computers relatively easy-to-recall names, such as `\\mypc` or `mypc.acme.com`, and let WINS or DNS remember the computers' more cryptic IP addresses. Thus, you can use the friendlier names for all your network usage and administration tasks and rarely need to deal directly with IP addresses. For example, when you type

```
net view \\mypc
```

at a command line, your system automatically knows to ask a WINS server to convert the name `\\mypc` to the computer's IP address, or more technically, to resolve the name `\\mypc`.

Every NT 4.0 or NT 3.51 system running TCP/IP (or for that matter, any Windows 9x or Windows for Workgroups—WFW—3.11 system) needs to know the IP address of a WINS server that the system can count on to resolve names. That WINS server, which is called the system's primary WINS server, performs two functions: system registration and name resolution. In my opinion, the fact that WINS uses the same server for both system registration and name resolution is a weakness of NT. Let me explain why.

WINS can't resolve names to IP addresses unless it knows the names and addresses of the systems on your network. Where does it get that information? Every system that relies on WINS for name resolution must contribute to WINS's database of names and addresses. In other words, each WINS-using client system must register its information with a WINS server: "Hi, WINS. This is `\\mypc`, and I'm at address 10.4.15.11. Please tell anyone who is looking for me to go to that address."

But systems don't register just once. WINS servers remember systems' address information for only a few days, depending on how the administrator set up the WINS server. So systems reregister with WINS regularly—every time they boot and whenever they renew their DHCP leases, for example. Consequently, a WINS server is usually busy in the morning because it needs to accept a lot of registrations from machines that employees boot in preparation for the workday.

You'd think that you could lighten the registration load by increasing the number of WINS servers on your network and assigning fewer workstations to each one. However, that's not a good idea. All those WINS servers need to share the names and addresses that they learn, combine them, boil them down to one unified database, then distribute that database to all WINS servers so that each WINS server can resolve every name in the network. This boiling-down and distribution process is called replication. If you install more than a dozen or so WINS servers in your enterprise, the process can go awry, leaving a corrupted and useless database. As a result, Microsoft has always said that you shouldn't have more than 15 WINS servers in your entire company, no matter how large or geographically distributed the company is.

But remember that WINS servers serve two functions. In addition to registering systems, WINS servers resolve names. Clearly, you'd like your network to respond quickly to name queries. One way to achieve the desired response time might be to have a large number of WINS servers, each with a copy of the WINS database. Each WINS server might serve only a hundred or so workstations and therefore could respond speedily to name-resolution queries. Unfortunately, that approach doesn't work because it also gives you many registration-taking WINS servers and thus opens the door to potentially unacceptable replication difficulties.

## The Differences in DDNS

Let's imagine a better approach. What if you had a small set of WINS servers that only took registrations and a much larger set of WINS servers that only responded to queries? The few registration-taking WINS servers could keep a nice, clean database and would see to it that the query-only WINS servers would always have the most up-to-date database. You'd have to change things around a bit on the client side because you'd have to tell workstations the names of two WINS servers. For example, you'd tell \\mypc, "When you need to register or reregister, go to WINS server A, and when you need to resolve a name, go to WINS server B." If you think that system seems better, then you're going to like dynamic DNS (DDNS) under Win2K. With DDNS, your workstation registers its name and IP address with a particular DNS server but might turn to a different DNS server to resolve names.

In the standard DDNS world, only one DNS server accepts all the registrations for your entire domain. (If you're running Win2K DNS servers, you can change that, but let's restrict this discussion to standard DDNS implementations.) That DNS server, which is called the primary DNS server for its domain, stores all the registered names for the entire domain in a file called a zone file. (Note to the DNS techies reading this chapter: Because of space limitations, I'm blurring the line between zone and domain.)

To provide DNS servers that can resolve names for your domain, what do you need? The easiest answer is that you need to set up some DNS servers. They don't need to be running DNS server software that supports DDNS. Nor do they need to possess a local copy of your domain's zone file, because unlike WINS servers, DNS servers can resolve names for domains they've never heard of. DNS servers simply search the hierarchy of DNS names and servers to find a server that

can resolve the name query, then obtain that server's assistance to resolve the name. For good measure, DNS servers remember (i.e., cache) each name query request. As a result, if you ask a DNS server to resolve the same name in the near future, the DNS server has the information it needs to immediately resolve the name. DNS servers that only resolve name queries and aren't responsible for any domains are called caching-only servers.

Under WINS, your workstation needs to reregister with the WINS server that resolves names. Under Win2K DDNS, the DNS server specified as your workstation's preferred DNS server only resolves names. When your workstation needs to reregister with the primary DNS server on your domain, your workstation tells its preferred DNS server, "I'm part of the acme.com domain. Please tell me who is the primary DNS server for acme.com." In more technical terms, your workstation asks its local caching-only DNS server to retrieve the Start Of Authority (SOA) record for the acme.com domain. The SOA record contains several pieces of information about the acme.com domain, including the DNS name (not the IP address) of acme.com's primary DNS server. Armed with that information, your workstation knows who to contact to reregister.

So DNS seems to afford us greater flexibility in offering as many name-resolution servers as we could want. But what about the registration side? It seems that every system in the enterprise will have to reregister every day on the domain's primary DNS server. Won't that requirement cause a problem?

Potentially, yes. But that problem might not be severe. For one thing, reregistration might not be necessary as long as your workstation's IP address doesn't change. Unlike WINS records, old DDNS records don't expire and disappear by default (although you can change that behavior if you want to). So if the primary DDNS server is too busy to reregister your workstation, you probably won't notice because your workstation already has a record in DNS. But for overloaded primary DDNS servers, DDNS offers an option called Active Directory (AD)-integrated zones. This option lets you spread the registration-taking responsibility among all the AD domain controllers (DCs), which all act as DDNS servers.

## Chapter 4

# DNS and Active Directory

—by *Mark Minasi*

Active Directory (AD) is a nice bit of technology—particularly for a version 1.0 technology—that has grabbed many headlines in the Windows 2000-related literature in the past couple of years. But to work correctly, AD needs its own directory: the directory of servers and workstations that you know as DNS. Before you start your AD planning, you need to do your DNS homework—or the best-planned AD will run badly.

Most of us learned the basics of DNS—DNS zones, Start of Authority (SOA), Name Server (NS), mail exchanger (MX), host (A) records, primary and secondary servers, and the like—under Windows NT 4.0, perhaps by setting up a simple DNS server. But to build a sturdy DNS structure that supports AD with style, you need to know more. For example, should you use the DNS server that ships with Win2K or a non-Microsoft DNS server, such as the almost ubiquitous Berkeley Internet Name Domain (BIND) software that runs on so many UNIX and Linux boxes? Where should you put DNS forwarders and slaves—and more basically, what do those terms mean? What's split-brain DNS? And how can you make AD coexist with an existing DNS infrastructure?

## Choosing a DNS Server

Don't assume that you'd be foolish to host your AD on a BIND (or Lucent Technologies' Lucent QIP or other third-party) DNS server. BIND is quite capable of working with AD—and has been since Win2K's inception.

To convince myself that AD and BIND get along, I spent one Saturday blowing up my existing three-domain AD forest and rebuilding it without using Microsoft DNS servers. Instead, I used MandrakeSoft's Linux-Mandrake 7.1 and the version of BIND available at the time. In addition to going smoothly, the process of building the AD forest seemed to go a bit faster than it did with a Win2K-based DNS server. I used the BIND-based DNS structure to support my AD for several months thereafter, and during that time I didn't hit a single snag.

I'm not implying that you shouldn't consider using Win2K's DNS server. That server offers an attractive, if nonstandard, option called AD-integrated zones. AD-integrated zones provide two things: multimaster primary zones and secure dynamic DNS (DDNS) updates. (For more information about Win2K's standard and AD-integrated zone files, see the sidebar "DNS Zone Types.") Win2K uses DDNS in the same way that NT 4.0 uses WINS: as a repository of computer names and IP addresses as well as a central directory of server types. When an NT 4.0 workstation wants to find a domain controller (DC) to help the workstation log on to an NT 4.0 domain, the workstation goes to WINS to find a DC. In contrast, a Win2K Professional workstation that's a member of an AD domain queries DDNS to find a DC. The DDNS server must then gather the names and roles of the machines in your domain. Win2K-based systems assist DDNS in that function by registering their names with a DDNS server. Thus, Win2K machines essentially introduce themselves to the DDNS server and augment its database of names and addresses.

## DNS Zone Types

—by *Robert McIntosh*

DNS plays an important role in creating an effective Windows 2000 Active Directory (AD) implementation. AD requires DNS and uses it for name resolution and, with the help of a new Resource Record (RR) type called SRV records, for service location. Because AD relies on DNS for these services, Win2K offers a more scalable and efficient solution than Windows NT 4.0, which uses WINS. A DNS database known as a zone file contains RRs to link host names with their corresponding IP addresses. Win2K DNS supports two kinds of zone files, standard and AD-integrated.

### Standard Zone Files

Standard zone files are traditional DNS zone files. To use standard zone files, you create a zone on the DNS server that you plan to use to perform DNS database administration. This server becomes the primary zone server where all updates, such as RR additions or deletions, occur. When you create a DNS server to function as a secondary zone server, you specify the name or IP address of the primary zone server that will provide a copy of the zone file. You can use secondary zone servers to provide load balancing and a certain degree of fault tolerance. Secondary zone servers provide only limited fault tolerance because they continue to respond to DNS queries; secondary zone servers can't perform any updates because they have only a read-only copy of the zone file. The primary zone server periodically replicates its zone file to the secondary zone server to ensure that the secondary zone server's copy is current. With earlier versions of Microsoft DNS, the primary zone server transfers a full copy of the zone file and overwrites the existing zone file on the secondary zone server. Win2K DNS supports Incremental Zone Transfers, which means that the primary zone server sends only changes that have occurred to the zone file since the last replication.

### AD-Integrated Zone Files

With Win2K, you can also use AD-integrated zone files to incorporate zone file information into AD. With this approach, DNS uses AD for zone file storage and replication, which has advantages over standard zone types. Because the AD-integrated zone file process uses AD's replication service, you don't need to configure a separate replication topology. AD-integrated zone files also eliminate the single point of failure that arises when a standard primary server goes down. With AD's multimaster approach, you can make DNS changes at any domain controller (DC), and the changes automatically replicate to the other DCs in the domain according to AD's default replication topology.

Continued on page 25

**DNS Zone Types** *continued*

Although both zone types support the dynamic update protocol, dynamic DNS (DDNS), only AD-integrated zones support secure dynamic updates, which let you control who can update DNS and reserve a particular name for a specific server to use.

Keep in mind is that AD-integrated zone files don't replicate between domains. This limitation follows the usual AD replication model in that most information replicates only to other DCs in the same domain. This issue is especially confusing because the Microsoft Management Console (MMC) DNS snap-in lets you create zones in multiple domains with the same name.

**Creating Zones and Changing Zone Types**

To create a new zone, right-click either the Forward or Reverse look up folder in the MMC DNS snap-in, and chose New Zone. A wizard appears and asks what type of zone you want to create. However, note that the option to create an AD-integrated zone won't appear if you haven't already run Dcpromo. In such cases, you can create a standard zone and change it after you create your AD by right-clicking the zone name in the DNS snap in and choosing Properties. You can follow this same procedure whenever you need to change zone types.

But which DDNS server do Win2K-based systems introduce themselves to? In a standard DDNS system, such as a computer that's running either BIND or Win2K's DDNS server in primary zone mode rather than AD-integrated mode, only one DDNS server—the primary DDNS server—can accept registrations. Consider a worldwide enterprise that has thousands of machines around the globe seeking to reregister every morning: All those machines need to reregister with the same computer—the only computer that can accept those registrations. In contrast, an AD-integrated zone lets you designate any or all DCs to do double duty as primary DDNS servers and accept registrations. Machines can then register DNS names with local DCs, reducing WAN traffic.

However, registration is crucial only for servers and DCs. So, you could alternatively attack the registering-over-the-WAN problem by telling Win2K Pro boxes not to bother registering. (You make this change on the DNS tab of the TCP/IP Advanced Properties page.) Furthermore, DDNS differs from WINS in that DDNS registrations don't expire in a few days—by default, DDNS registrations stay in a DDNS zone forever. So, even if a machine can't reregister every morning, DDNS will contain a record for that machine, assuming that the machine registered at least once in the past.

AD-integrated zones have another feature that you might find attractive—secured DDNS registrations. If you use the default options to set up a DDNS zone on a BIND or Win2K DNS server, anyone who can connect to that server can register a machine in that zone. BIND lets you prevent registrations from machines outside certain subnets: When you list the subnets that the BIND server should accept registrations from, the server ignores all other machines. An AD-integrated zone lets you restrict DDNS registrations in a different way: You can require registering machines to log on to the AD domain before they register. This approach is a bit easier to set up than the

BIND approach—simply click the General tab of a zone's Properties page, click the Allow dynamic updates? drop-down list, select Only secure updates, and you're finished.

## Using Caching-Only Servers

Consider how you'll place DNS servers in your intranet to serve your users' needs. Many DNS servers exist to hold copies of an organization's zone files, as you've seen if you've ever set up a DNS server. But a lot of DNS servers hold no zones, living only to resolve names, whether on the Internet ("What is the IP address of <http://www.microsoft.com>?") or on your intranet ("Where is the nearest DC for [acme.com](http://acme.com)?"). Such a DNS server is called a caching-only server. After you set up a zoneless DNS server, you can see a reference to its caching-only nature in the event log (event ID 708).

A caching-only server's strength lies, as its name implies, in the fact that DNS servers remember the results of previous resolutions. For example, if someone in your office points his or her Web browser to <http://www.cnn.com>, the Web browser asks its preferred local DNS server to find the IP address of <http://www.cnn.com> from CNN.com's DNS server. The preferred local DNS server goes out on the Internet to get that information, and that process takes time. But the second person to ask the local DNS server for <http://www.cnn.com>'s IP address gets a nearly immediate response because the server resolves the name out of its cache rather than turning again to the Internet for the answer.

However, the local DNS server will eventually return to CNN.com's DNS server to determine whether <http://www.cnn.com>'s IP address has changed. The reason for the return trip is that when the CNN.com DNS server responded to the initial query, the response included not only <http://www.cnn.com>'s IP address but also the amount of time that the local DNS server should cache that IP address. That amount of time is called the Time to Live (TTL). All responses to DNS resolution requests contain a TTL. After the TTL expires, a new query causes the local DNS server to return to the Internet to resolve the name.

## Centralizing Cache with Forwarders

You can see that caching DNS resolutions is a great idea. If CNN.com's TTL were 2 days long, a local caching-only DNS server could resolve <http://www.cnn.com> for 2 days before having to recontact CNN.com's DNS server.

Now, consider the effects of adding a second local caching-only DNS server—let's call the first local DNS server DNS1 and the second server DNS2. When DNS1 learns something, it doesn't share that information with DNS2. For example, if someone were to ask DNS2 for <http://www.cnn.com>'s IP address, DNS2 must get the information from CNN.com's DNS server even if DNS1 has cached that IP address. In other words, DNS1 and DNS2 don't share a cache.

But if you use a forwarder, servers can share a cache. The idea with a forwarder is this: You set up some local caching-only DNS servers as usual to serve the name-resolution needs of workstations and other servers. Then, you set up another server that acts as a sort of DNS server for these DNS servers. In DNS terms, this server is a forwarder.

When one of the local caching-only DNS servers gets a request to resolve a name that it doesn't already have in its cache, that DNS server doesn't go to the Internet to resolve the name; instead, the server asks the forwarder to resolve the name. If the forwarder—which is on the same LAN as the local caching-only DNS server and can communicate with that server at high speeds—

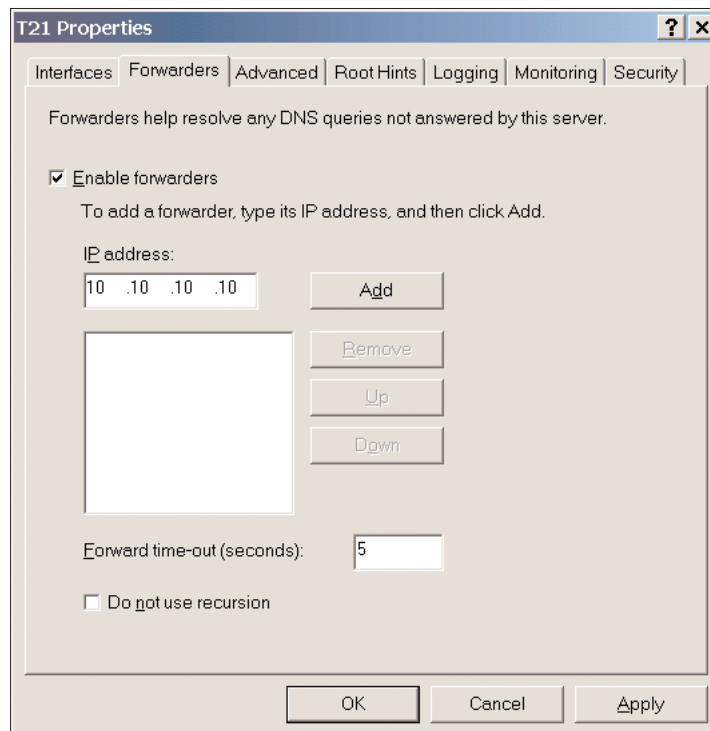
knows the answer, the forwarder can quickly answer the local caching DNS server's question. If the forwarder doesn't know the answer, the forwarder obtains the answer from the Internet.

Suppose DNS1 serves the first person to ask for the IP address for `http://www.cnn.com`. Because DNS1 doesn't know the answer, it asks the forwarder. The forwarder doesn't know, either, because no one has asked the question before, so the forwarder goes to the Internet for the answer. After the forwarder finds `http://www.cnn.com`'s IP address, it caches the address, then sends that information to DNS1.

Now, suppose the second person who wants the IP address for `http://www.cnn.com` is connected to DNS2. DNS2 can't resolve the address from its cache, so DNS2 asks the forwarder. This time, the forwarder knows `http://www.cnn.com`'s IP address and satisfies DNS2's request at LAN speed.

To tell a Win2K DNS server to use another DNS server as a forwarder, simply open the Microsoft Management Console (MMC) DNS snap-in and right-click the icon that represents the DNS server that you want to act as a forwarder. Choose Properties, and you'll see a Forwarders tab, which Figure 1 shows. The Forwarders tab lets you specify one or more forwarders and a timeout value.

**Figure 1:**  
*The MMC DNS snap-in's Forwarders tab*



Why would you want a timeout value? Well, if your forwarder stops working, DNS1 and DNS2 won't be able to get the answers to new name-resolution questions. DNS handles that eventuality with a timeout. If a local caching DNS server such as DNS1 or DNS2 asks the forwarder a question and the forwarder doesn't respond within a certain length of time, the local caching DNS server goes to the Internet and resolves the name on its own. (As I explain later, that action is a potential liability.)

## Using Split-Brain DNS

Take a look at a DNS zone in an AD domain, and you'll notice two things. First, AD stores a lot of stuff in DNS. Second, much of that stuff is information that you wouldn't want anyone outside the organization to see. In particular, a zone in an AD domain contains the names and addresses of your DCs, and you probably don't want the world to have that information.

One way to avoid running a too-visible network is to keep two sets of DNS books: one DNS zone that the outside world can see and another DNS zone that only the internal organization can see. Some people call this arrangement split-brain DNS. Here's how it works.

When you ask your preferred DNS server to resolve a name for you, the server first looks in its cache. Then, if the name isn't in the cache and the DNS server holds any zones, it looks in those zones to try to resolve your request. The server goes to a forwarder or to the Internet only if it can't answer the request from its cache or its zones. The key to split-brain DNS is that the DNS server favors the information in its zones over information it can find on the Internet.

Let's look at an example to see how to set up these two sets of books. Suppose we have a domain named `acme.com`. On that domain, we have a few DNS addresses that we want the outside world to be able to see—two DNS servers (`ns1.acme.com` and `ns2.acme.com`), a mail server (`mail.acme.com`), and a Web server (`www.acme.com`). Our externally visible `acme.com` zone, then, is slim—fewer than a dozen records, all of which refer to routable IP addresses accessible through the public Internet. We create that zone file and put it on our two externally visible DNS servers, `ns1.acme.com` and `ns2.acme.com`. (We might not even run our own DNS servers—for a zone that small, many firms simply let their ISP host the files.) Network Solutions has the addresses of `ns1.acme.com` and `ns2.acme.com` in its database of all DNS servers, and any DNS server outside of Acme—the kind that a user on the Internet might query—will need to ask Network Solutions' DNS servers where to find the `acme.com` servers. Consequently, the only servers that outsiders will see are `ns1.acme.com` and `ns2.acme.com`.

Inside Acme, however, we have an intranet built on nonroutable addresses—perhaps a `10.x.x.x` network—that uses some kind of port address translation scheme to connect to the Internet. Anyone inside Acme can initiate communication to the Internet, but people on the Internet can't initiate communication to our internal `10.x.x.x` addresses. (To keep this example simple, the port address translation is the only “firewall” that we'll imagine for Acme.)

On the intranet, we set up a group of DNS servers that act as preferred DNS servers for other machines on the intranet. Those DNS servers use the external `acme.com` DNS servers as forwarders, enabling people within the intranet to resolve names and surf the Internet. The internal DNS servers aren't simply caching-only DNS servers; those internal servers also hold a copy of an `acme.com` zone. However, that zone isn't the externally visible `acme.com` zone that `ns1.acme.com` and `ns2.acme.com` hold. Rather, the internal DNS servers hold an `acme.com` zone that AD and

DDNS built. If that internally visible acme.com zone is an AD-integrated zone, then the internal DNS servers are Win2K DCs for the acme.com AD domain.

If you've never set up a split-brain DNS structure, you should take a minute to review and think about what's going on. None of the systems within the Acme intranet use the external DNS servers as their preferred DNS-name resolvers—rather, the machines on the intranet look to the intranet's DNS servers to resolve names. Those internal servers hold copies of a zone called acme.com. But the acme.com zone that the internal servers hold is different from the acme.com zone that the outside world sees (although the zone on the internal servers presumably contains the same records that the external servers' zone contains—records that point to the Web and perhaps to mail servers). Instead, the internal zone contains the information that Win2K systems need to locate DCs.

The outside world will never find or refer to those intranet DNS servers because Network Solutions' DNS servers don't know about Acme's intranet DNS servers. So, an attempt to resolve an acme.com name from the Internet never reaches an intranet DNS server. Even if a user from outside Acme were to attempt to use an internal DNS server, that attempt would fail because the internal server has a nonroutable IP address.

## Using Slave Servers to Protect Intranet Servers

Before leaving the Acme example, I want to note a potential security problem. I said that the intranet DNS servers use the external DNS servers as forwarders. But recall what happens when a forwarder doesn't respond quickly enough to a name-resolution request—the intranet DNS server searches the Internet's DNS servers to try to find the answer. Security experts say that action results in a potentially troublesome hole in security. Simply put, your DNS server could end up connecting to a computer that's masquerading as a DNS server. That false DNS server could exploit the connection to your intranet DNS server to do various kinds of mischief.

To avoid this exposure, you can tell your intranet DNS servers that if the external DNS servers timeout without resolving a name, the intranet servers must not attempt to resolve the name on their own. To configure your internal servers, go to the Forwarders tab in the MMC DNS snap-in and select the *Do not use recursion check box*. You then have what Microsoft calls a *slave server*.

## Fitting into an Existing DNS Structure

What if Acme already has a DNS structure in place—perhaps some non-Win2K computers run by people who've been managing the DNS structure for years without having to worry much about Win2K and NT? Those folks might not be happy about having AD put a lot of information in their zones. Or maybe Acme's existing DNS infrastructure doesn't support DDNS or Internet Engineering Task Force (IETF) Request for Comments (RFC) 2782 SRV records, making the infrastructure unacceptable for AD. If you're faced with an AD-unfriendly infrastructure, what can you do?

The simplest answer is that you can create a subdomain. Instead of giving the AD domain the same name as the organization's top-level domain (TLD)—acme.com, for example—you can create a subdomain named something like win2k.acme.com or ds.acme.com, where ds stands for directory service. The benefit of this approach is that it doesn't affect the TLD's zone very much. Creating a child domain, such as ds.acme.com, in acme.com requires only a few records in the acme.com zone file: an NS and an A record for each DNS server in the subdomain. The NS record points to a Win2K system that acts as the DNS server for the ds.acme.com subdomain.

The bottom line is that AD requires a well-built DNS underpinning to work correctly. But the techniques of DNS design aren't complex; they're just new to most of us. If you use the points in this chapter to build your DNS structure, you'll be well on your way to a sturdy AD domain.

## Uprooting DNS

Suppose you've set up a Win2K-based DNS server on your network. To configure it, you right-click within the MMC DNS snap-in window, then choose Properties. Then, some strange things happen. You can't access the Forwarders tab because it's grayed out, so you can't tell the DNS server to use a forwarder. The Root Hints tab is also grayed out, so you can't change the list of root servers that your DNS server knows about. What's going on?

For some reason, Win2K configured your DNS server as a private root server. Recall that DNS is a hierarchy of names, as in the Fully Qualified Domain Name (FQDN) `www.minasi.com`.—and yes, that period at the end of the name is deliberate: No DNS name is complete without a period (.) at its end. This name reflects that a machine named `www` is in a domain named `minasi`, which is a child domain of the domain named `com`, which itself is a child domain of the topmost domain of all—the root domain. But we don't spell the root's name out; we represent it with a period. An actual DNS server—13 of them, in fact—contains the DNS records for the root domain. And that domain is queried quite a bit—you typically can't find other domains without first finding the root.

Win2K's DNS server knows that. The first time that you start up a Win2K-based DNS server, it appears to look for the root domain. If the server doesn't find a root domain, it seems—I say “seems” because this behavior appears to be inconsistent—to respond by saying “if I can't find a root, then I must be the only DNS server in the universe” and creating a root domain on itself, in effect disconnecting from the public DNS hierarchy. The server solipsistically decides that it's the authority and that trying to find another DNS server is pointless. Until you cure the server of its delusion, it won't be much good.

Well, let me rephrase that a bit: At times, you might like to have a DNS server unconnected to the public DNS hierarchy. A very secure network, completely disconnected from the Internet, wouldn't need to be able to search the public DNS hierarchy. But in most cases, this situation isn't what you want.

Telling a DNS server that it isn't the root shouldn't be difficult, but it can be frustrating if you don't know how. Here are three ways to introduce your server to the rest of the world. I explain all three so that I can also offer some insight into how to do a bit of under-the-hood work on DNS.

All the methods share the same basic approach: They delete the root domain from your DNS server. The command-line method uses a command called `dnscmd.exe` from the Win2K Server Tools add-on. You'll need to install the command; you'll find it in the `\support\tools` folder on the Win2K Server CD-ROM. Then, you can delete the root from your DNS server by typing

```
dnscmd /zonedelete .
```

The period at the end of the command is the name of the zone to delete. You must use `Dnscmd's /dsdel` option if the zone you're deleting is an AD-integrated zone (although I'm not sure how you'd end up with an AD-integrated root domain). In the command, case doesn't seem to matter.

After you execute the command, I recommend restarting the DNS service. Then, open the DNS snap-in, and you'll see that the Forwarders and Root Hints tabs on the server's Properties page are no longer grayed out.

You can also deroot a DNS server from the DNS snap-in. Open the Forward Lookup Zones folder, and you'll see a folder whose name is simply ".", which, as you now know, is the root domain. Right-click that folder, choose Properties, and check that the zone type is Primary rather than AD-integrated; if the zone is AD-integrated, click Change next to the type and convert the zone to a standard primary zone. Then, in the Forward Lookup Zones folder, right-click the root zone and click Delete. Close the DNS snap-in, restart the DNS Server service, then reopen the DNS snap-in. The Forwarders and Root Hints tabs will no longer be grayed out.

As far as I know, the third uprooting approach works only on a standard primary zone. First, stop the DNS Server service, either from the GUI or by typing

```
net stop "dns server"
```

at the command line. Then, edit the registry to remove the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\DNS\Zones\. subkey (the final period is part of the subkey). Restart the DNS service, and you're back in the public DNS hierarchy.

## Chapter 5

# Navigating Name Resolution

—by *Sean Daily*

If you asked me, as a network consultant, to choose the most problematic area of Microsoft Windows networking, my unblinking answer would be name resolution. You can directly relate ailments such as sluggish performance, client computers' or applications' inability to connect to servers, incomplete browse lists, and obscure error messages to problems resolving computers' names to their IP addresses. To help you overcome these maladies, I offer a name-resolution primer, discuss some of the lesser-known causes behind name-resolution problems, and resolve a common name-resolution misunderstanding.

## An Equal Opportunity Annoyer

In the Internet's DNS-driven namespace, name resolution is fairly straightforward. To resolve names, clients query DNS servers, which either provide the IP address of the host in question or inform the client that no such name exists—a generally transparent process. Not so with Windows-based networks' name-resolution process, which involves additional layers and services, including WINS servers and clients, broadcasting, and static name-to-IP address mapping files (i.e., HOSTS and LMHOSTS files). These additional name-resolution methods complicate the process and make diagnosing and resolving problems trickier. Before I jump into the complicated mechanics, I want to cover the fundamentals of name resolution in a TCP/IP environment.

## Resolution Rudiments

Unlike protocols that rely exclusively on broadcasting (e.g., IPX/SPX, NetBEUI) to resolve names, TCP/IP in a Windows environment can also use point-to-point name server queries. Point-to-point queries are preferable because they're more reliable, can easily cross routers, and use less network bandwidth than broadcast queries. (A point-to-point query is like a friend calling you on the telephone to ask a question, whereas broadcasting is like a friend standing in front of your house yelling into a megaphone to ask a question. In the former case, the communication involves only your friend and you; in the latter case, all your neighbors become involved.)

## The Four Node Types

Clients of all Windows OSs—except Windows 2000—require the presence of the NetBIOS session layer with a network transport protocol. (Win2K doesn't rely exclusively on NetBIOS to support networking activities: Win2K's TCP/IP implementation can use DNS to locate network servers and services.) So, the OSs' TCP/IP protocol stacks support a modified form of TCP/IP called NetBIOS over TCP/IP (NetBT). Windows clients that use NetBT (i.e., WINS clients) can attempt name resolution by using one of four NetBT node types: b-node, p-node, m-node, and h-node.

### **B-node**

B-node clients use broadcasts to register their names and resolve the names of other machines on the network. B-node is undesirable for most networks because IP routers typically can't forward broadcasts, so b-node limits a client's scope of resolution to one IP subnet (i.e., network segment). And because b-node relies on network-saturating broadcasts, it uses more bandwidth than point-to-point methods. B-node is the default for Windows clients that don't have a configured NetBIOS Name Server (NBNS—the generic name for a WINS server, which is typically a Win2K- or Windows NT-based server). NT also supports a slightly altered form of b-node called *modified b-node*. A modified-b-node client first checks its local name cache for a name-to-address mapping. If the client doesn't find the address in memory, it resorts to broadcasting. If neither of those methods works, the client checks the local LMHOSTS file (if it exists) in the `\%systemroot%\system32\drivers\etc` folder on Win2K or NT systems or in the Windows folder on Windows 9x systems.

### **P-node**

P-node clients register their names with a WINS server and use that server to resolve names. P-node has a major drawback: If the server can't resolve a name or if you've configured the client with incorrect name-server addresses, the client can't log on to the network because it can't discover the name of a domain controller (DC) to process the logon request.

### **M-node**

M-node resolution is one of two hybrid node types. When attempting to resolve names, m-node clients use b-node-style broadcasts then point-to-point WINS server (i.e., p-node) queries, in that order. M-node is optimal for WAN subnets that have no local name server and connect to the WAN by low-bandwidth links. In these cases, using broadcasts to resolve names helps minimize traffic on the WAN links.

### **H-node**

H-node resolution is the default for Windows clients that you've configured (either manually or with information from a DHCP server) to register with one or more WINS servers. H-node is essentially the opposite of m-node: Clients register their names directly with WINS servers and use these servers first when attempting to resolve names. If the name server queries fail, the client then uses b-node-style broadcasts.

How do you determine a WINS client's node type? By default, a client uses b-node unless you configure it to register with at least one WINS server, in which case the client will use h-node. However, you can override this behavior. For example, you can use a DHCP scope option (i.e., a configuration parameter that passes to a DHCP client) to configure clients to use a specific node type. (You can also use a registry entry to manually configure the WINS node type setting on most Windows versions. However, I don't recommend this method because the manual registry configuration overrides even a dynamic DHCP-assigned node type, making this method less flexible.)

Because a majority of NT networks use WINS servers, a majority of Windows-based LAN clients on these networks use h-node resolution. If neither point-to-point nor broadcasting is successful, h-node WINS clients then use other methods. H-node uses the following progression to attempt name resolution:

1. Check whether the queried name belongs to the local machine.
2. Check the local name cache, which by default retains resolved names for 10 minutes.
3. Direct a point-to-point query to the configured WINS server or servers.
4. Use a broadcast query.
5. Use a local LMHOSTS file, if one exists.
6. Use a local HOSTS file, if one exists. (A HOSTS file resides in the `\%systemroot%\system32\drivers\etc` folder on Win2K or NT systems or in the Windows folder on Win9x systems.)
7. Issue a point-to-point query to any configured DNS server or servers. In NT, h-node WINS clients will attempt this step only when you select the *Use DNS for Windows name resolution* check box in the TCP/IP Configuration dialog box. This setting maps to the EnableDNS registry subkey of type REG\_DWORD under the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\NetBT\Parameters subkey in NT or the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\VXD\MSTCP subkey in Win9x. Despite its name, the EnableDNS subkey controls only whether a client uses DNS servers during NetBT name resolution and doesn't affect the use of DNS for other purposes (e.g., resolving Web site names in a browser). By default, NT disables the setting (i.e., sets the value to 0), whereas Win9x enables the setting (i.e., sets the value to 1).

Now you understand some of the basics. But you still need to examine why name resolution can cause networking problems.

## DNS Where You Least Expect It

DNS plays a much larger role in name resolution than the previous method order implies. H-node WINS clients query DNS servers only after all other name-resolution attempts have failed and only if you've enabled the client to use DNS for Windows name resolution. However, h-node WINS clients attempt to use a DNS server as if it were a WINS server: The client uses a WINS-style NetBIOS name query, rather than a DNS-client name query, to send a name-resolution request to the DNS server. As a result, unless your DNS server happens to be your WINS server and has a registration in its database for the queried name, this h-node attempt is likely to fail.

The NetBT h-node resolution process isn't the only time that clients query DNS servers during a name-resolution attempt. If you've enabled DNS on a client and configured the client to use one or more DNS servers, the client will use DNS client (aka Domain Name Resolver—DNR) queries through the standard UDP port 53 to send name queries to DNS servers. The client will send these DNS-centric queries after checking for a local HOSTS file and before any NetBT name-resolution attempts. To create a Fully Qualified Domain Name (FQDN—for example, `myserver.mybiz.com`) for the query, the client will append any default names that you've configured in the DNS section of its TCP/IP configuration dialog box to the queried name. (I used a network-monitoring tool to observe this behavior. For information about how to view client name resolution and other network traffic, see the sidebar "Getting Under the Hood with a Network Monitor.") The following is a complete and accurate order in which Windows LAN clients attempt to resolve names:

1. Check whether the queried name belongs to the local machine.
2. Use a local HOSTS file if one exists.

## Getting Under the Hood with a Network Monitor

When you want to observe name-resolution behavior on your Windows network, I recommend that you use a network-monitoring product, such as Microsoft Network Monitor for Windows NT, Network Monitor for Microsoft Systems Management Server (SMS), or a third-party product. Figure A shows an example network-monitoring session, which displays the results of a WINS client using the Ping command for a name query. Frames 1 through 8 show that the network traffic from the client starts with a series of DNS queries to each of the clients' configured DNS servers, each of which fails. Frame 9 shows that the client then sends the query to the first configured WINS server, which responds with the queried name's resolved IP address, as Frame 10 shows. Frames 11 through 13 show the Internet Control Message Protocol (ICMP) traffic that the Ping command generates between the two hosts.

**Figure A:**

*Using a network-monitoring tool to observe name-resolution behavior*

The screenshot shows the NetXRay interface with a list of 13 frames. Frame 9 is highlighted, showing a NetBios-NS C=Name Query-Questi. Below the list, the details for the selected frame are expanded to show the Domain Name Service (DNS) structure, including the Header Section (Identifier: 1, Flags: Req, Query, Non-Auth, Recu Desr, Recu Not Ava, RCode=No Error) and the Question Section (Type=A, Class=IN, exchangeserver).

No.	St...	Source Address	Dest Address	Layer	Summary
1	Ok	192.168.0.98	208.201.224.11	DNS	Query Question:Type=
2	Ok	192.168.0.98	208.201.224.33	DNS	Query Question:Type=
3	Ok	192.168.0.98	208.201.224.11	DNS	Query Question:Type=
4	Ok	192.168.0.98	208.201.224.33	DNS	Query Question:Type=
5	Ok	192.168.0.98	208.201.224.11	DNS	Query Question:Type=
6	Ok	192.168.0.98	208.201.224.33	DNS	Query Question:Type=
7	Ok	192.168.0.98	208.201.224.11	DNS	Query Question:Type=
8	Ok	192.168.0.98	208.201.224.33	DNS	Query Question:Type=
9	Ok	192.168.0.98	192.168.0.1	NetBios-NS	C=Name Query-Questi
10	Ok	192.168.0.1	192.168.0.98	NetBios-NS	R=Pos-Name Query- Ar
11	Ok	192.168.0.98	192.168.0.4	ICMP	Type=Echo Request, I
12	Ok	192.168.0.4	192.168.0.98	ICMP	Type=Echo Reply, ID=
13	Ok	192.168.0.98	192.168.0.4	ICMP	Type=Echo Request, I

Domain Name Service

- HEADER SECTION:
  - Identifier: 1
  - Flags: Req, Query, Non-Auth, Recu Desr, Recu Not Ava, RCode=No Error
  - Section Entries: QDCount=1, ANCount=0, NSCount=0, ARCount=0
- QUESTION SECTION[1]: Type=A, Class=IN, exchangeserver

00000000: 20 53 52 43 00 00 44 45 53 54 00 00 08 00 45 00 | SRC...DEST...  
 00000010: 00 20 16 01 00 00 00 11 b3 40 00 00 00 00 00 00 | ...

3. Query any configured DNS servers using DNR queries on UDP port 53.
4. Use the appropriate NetBT node type resolution method, according to the client's NetBT node type (as I described earlier).

This list paints quite a different picture from the previous list. Most administrators I talk to are surprised to discover that Windows LAN clients consistently query DNS servers before they query WINS servers. (This misunderstanding is common because most Microsoft documenta-

tion and training materials that discuss NetBT resolution order neglect to mention these preliminary DNS queries.) Your WINS-enabled LAN clients are probably making an exorbitant number of failed DNS name-resolution queries on a regular basis. A DNS server that doesn't have the information simply responds that it can't resolve the queried name; this acknowledgment doesn't take much time, so users seldom notice any delay. However, if the client can't reach the DNS server or servers for some reason (e.g., you've misconfigured the server addresses, the servers aren't reachable from the client's IP address, the servers are down), the client must wait for successive DNS server queries to time out before it can attempt a NetBT method. Depending on the client OS and service pack revision level, this wait can range from 5 seconds to a whopping 2 minutes.

## PPTP and Other Problems

DNS-first behavior can prove problematic in several situations. For example, remote laptop clients with Ethernet connections and configured DNS-server addresses might experience long delays when they try to connect to registry-cached DNS server addresses configured through the primary connection (e.g., Ethernet connection, initial RAS modem connection). This situation can occur when the clients access a corporate LAN over a dial-up connection, but the corporate LAN's configuration doesn't permit access to the DNS servers configured on the primary connection. As a result, the client has to wait for each DNS server query to time out. These timeout periods are longer than WINS-associated timeouts because they take into account that the client might be connecting over the Internet—an unpredictable network compared with the private LAN or WAN that houses a WINS server. This problem occurs even with a RAS client that inherits reachable DNS servers from a RAS server because the RAS client places these additions at the end of its DNS server list and tries them only after it tries the original (i.e., unreachable) DNS servers.

RAS clients that use PPTP to dial in to a corporate VPN also experience the DNS-delay problem. By default, RAS uses the most recent connection as the client's default gateway. (RAS makes the most recent connection's default gateway-route metric lower than the previous connection's metric, as Figure 1 shows.) But PPTP connections are by nature secondary connections. As a result, the PPTP connection effectively orphans the primary (i.e., RAS) connection because all nonlocal traffic travels over the secondary (i.e., PPTP) connection by default. Unless the PPTP connection serves a network that permits the client to connect to the DNS servers that you configured through the primary connection, the client can't access those servers. However, the client doesn't know about this limitation, and the result is a significant slowdown on the client system as the client tries each DNS server and times out. Most DNS-enabled clients try to reach each configured DNS server four times before giving up. In a worst-case scenario, this timeout can cause a 2-minute delay.

A potential workaround for this problem is to clear the Use default gateway on remote network check box for the PPTP connection in the DUN client's phonebook entry. Only traffic destined for the PPTP-served network will pass over the PPTP DUN connection, and other traffic, such as DNS queries to Internet-based DNS servers, will pass successfully over the primary connection. This approach works well when the client connects to a single-LAN network. However, if the client uses a PPTP connection to connect to a WAN and needs to contact a machine that is across a router from the WAN's RAS server, this method won't work

because the client uses the default gateway on its primary connection instead of the default gateway on its PPTP connection.

**Figure 1:**  
*Viewing default gateway metrics*

```

C:\WINNT\System32\cmd.exe
C:\>route print
-----
Interface List
0x1 ..00 60 97 80 a4 41 ..MS TCP Loopback interface
0x2 ..00 00 00 00 00 00 ..3Com 3C90x Ethernet Adapter
0x3 ..00 01 d0 7a 42 80 ..NdisWan Adapter
-----
Active Routes:
Network Destination  Netmask          Gateway          Interface
0.0.0.0              0.0.0.0          192.168.0.34    192.168.0.34
0.0.0.0              0.0.0.0          208.201.247.129 208.201.247.222
127.0.0.0            255.0.0.0        127.0.0.1       127.0.0.1
192.168.0.0          255.255.255.0    192.168.0.34    192.168.0.34
192.168.0.34         255.255.255.255  127.0.0.1       127.0.0.1
208.201.247.128     255.255.255.128  208.201.247.222 208.201.247.222
208.201.247.222     255.255.255.255  127.0.0.1       127.0.0.1
208.201.247.255     255.255.255.255  208.201.247.222 208.201.247.222
209.204.134.130     255.255.255.255  208.201.247.129 208.201.247.222
224.0.0.0            224.0.0.0        192.168.0.34    192.168.0.34
224.0.0.0            224.0.0.0        208.201.247.222 208.201.247.222
255.255.255.255     255.255.255.255  208.201.247.222 208.201.247.222
  
```

## Exceptions to the Rule

While researching the DNS-delay problem, I discovered that the delays didn't occur consistently on all Windows clients. To find out why, I conducted an extensive battery of tests, using every version of Windows from Win95 to Win2K. I found that the long DNS timeouts were a problem with all versions of Windows except Win2K, NT 4.0 Service Pack 4 (SP4) and later, and Windows 98 Second Edition (Win98SE). As it turns out, Microsoft made significant changes in these versions to the way that clients perform DNS queries in the TCP/IP stacks. Specifically, Microsoft changed the algorithm that the DNR component uses to query DNS servers, thus effectively shortening the maximum possible timeout in worst-case scenarios. This change reduces client name-resolution times from 120 seconds to 19 seconds, with an average of 5 seconds in most cases (the exact number of seconds depends on the number of configured DNS servers and several other factors). Although the client still queries DNS servers first, the modifications effectively render the problem moot in most situations.

## But Wait—There's More

Understanding name-resolution methods and how DNS queries can cause problems for your clients is the first step on the road to solving name-resolution problems. But if neither of the earlier workarounds applies to your situation, you'll need to consider other options. Next, I'll show you how to take control of name resolution and improve the stability and performance of your network.

Even the savviest NT administrators don't fully understand Windows networks' name-resolution process. And who can blame them? In a Windows world riddled with multiple name-server standards, changeable client-resolution orders, self-destructive WINS servers, name-resolution-modifying service packs and patches, and a bevy of contradictory Microsoft articles, I can't imagine anyone avoiding problems. Your only hope is to understand Windows name-resolution method-

ology so that you can troubleshoot problems when they arise on your network. I've researched a few solutions that you can employ to ease your name-resolution woes.

## Good HOSTS

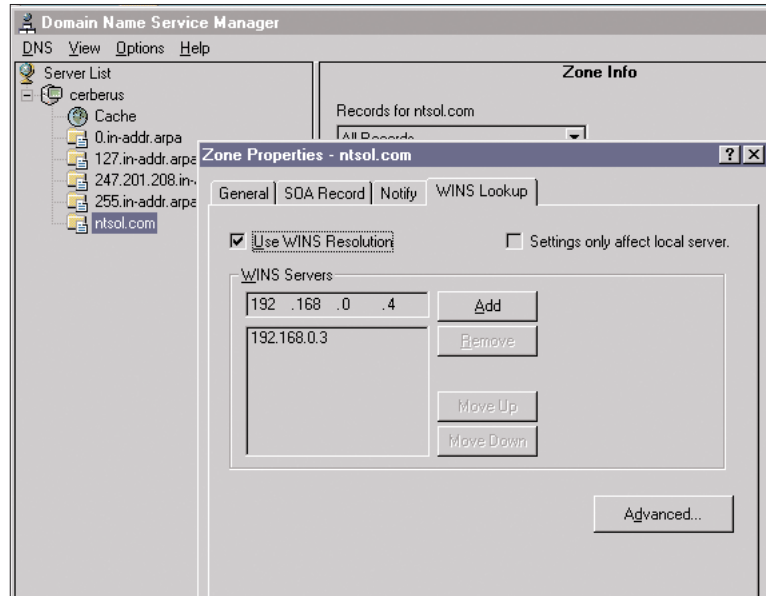
One solution that Microsoft recommends liberally in its articles is to use the client computer's HOSTS file, if one exists. A client always checks the HOSTS file before it checks DNS (i.e., HOSTS-file name resolution is second only to checking the client's local name cache). Therefore, you can avoid DNS-server-query timeouts if the HOSTS file contains the queried names. However, I don't like this solution because it requires you to manually deploy and update static text files on multiple remote machines as you add new machines to the network or change server names or addresses. The lone benefit of the HOSTS solution is that it provides the fastest possible name resolution. If speed is your top priority and you don't mind maintaining individual HOSTS files on multiple clients, this solution might be for you.

## Use WINS

Another solution is available for NT shops that run DNS servers. As I explained earlier, name-resolution delays can occur when a client can't connect to the DNS server (or servers) that acts as the client's primary connection. However, if that server is an NT DNS server that is accessible from both the internal network and the Internet, you can configure the server for all LAN and remote clients, go to DNS Manager, open a zone's Properties sheet, go to the WINS Lookup tab, and select the server's Use WINS Resolution option, which Figure 2 shows. When you take these steps, both internal and external users can talk to the server for name resolution, and the DNS server can respond to a client's dynamic-name query by querying WINS for the name. The Use WINS Resolution option integrates DNS with WINS and permits a dynamic form of DNS under NT 4.0. (This DNS configuration leverages WINS to provide dynamic capabilities but isn't the same as the dynamic DNS—DDNS—protocol that Win2K supports.)

This option might not be feasible for everyone. To take advantage of this solution, you must have an internal DNS server that is also accessible from the Internet, an IP addressing and firewall scheme that lets clients talk to the DNS server's IP address from within and from outside the LAN, and a willingness to maintain a DNS server or servers. Also, you might understandably have security concerns about a DNS server that is accessible to both the internal LAN and the Internet and about the possibility of obtaining information about internal IP hosts through the DNS server. You might be able to work around the second prerequisite (i.e., letting both internal and external clients access your DNS server's IP address) by entering two DNS server addresses on the client: one for the DNS server's external IP and one for its internal address. Thus, the worst-case scenario is that the client can't reach the first DNS server (e.g., when connected to the internal LAN through a RAS connection), so it moves on to the second address, which succeeds. In addition, you can mitigate the third prerequisite (i.e., maintaining a DNS server) by configuring the DNS server as a forwarder so that it simply passes queries on to Internet DNS servers such as those that your ISP maintains.

**Figure 2:**  
*Enabling NT's DNS server to query WINS*



## New Order

Another potential solution is to change the name-resolution order on a client. You can configure a client to use NetBT name-resolution methods and query WINS before it queries DNS. This way, you might avoid problems that stem from the client's default name-resolution behavior.

When I began to investigate this idea, I was encouraged to find several Microsoft articles that explained how to change the name-resolution order on clients running several Windows platforms. “How to Change Name Resolution Order on Windows 95 and Windows NT” (<http://support.microsoft.com/?kbid=139270>) describes two methods—one for NT and one for Win95—that let you manually control the order in which Windows attempts name resolutions. (For a list of name-resolution-related Microsoft articles, see “Name-Resolution Resources.”) I tried the Win95 method first. In Win95, you control name-resolution order through a group of four registry subkeys: LocalPriority, HostsPriority, DnsPriority, and NetbtPriority. (These subkeys are of type REG\_DWORD and reside under the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\VXD\MSTCP\ServiceProvider key.) The four subkeys control the relative priority of using the local name cache, the HOSTS file, DNS, and NetBT name-resolution methods (including WINS), respectively. The values can range from -32768 to 32767 (lower numbers specify a higher priority) with the following default values (in decimal):

- LocalPriority = 499
- HostsPriority = 500
- DnsPriority = 2000
- NetbtPriority = 2001

## Name-Resolution Resources

“Default Node Type for Microsoft Clients”  
<http://support.microsoft.com/?kbid=160177>

“DNS Client Does Not Try All Servers in DNS Service List”  
<http://support.microsoft.com/?kbid=195611>

“How to Change Name Resolution Order on Windows 95 and Windows NT”  
<http://support.microsoft.com/?kbid=139270>

“How to Disable NetBIOS Name Resolution on DNS”  
<http://support.microsoft.com/?kbid=137368>

“Microsoft TCP/IP Host Name Resolution Order”  
<http://support.microsoft.com/?kbid=172218>

“SP4 Changes DNS Name Resolution”  
<http://support.microsoft.com/?kbid=198550>

“Unable to Resolve NetBIOS Names Through PPTP Connection”  
<http://support.microsoft.com/?kbid=176321>

“Windows 95 ServiceProvider Priority Values Not Applied”  
<http://support.microsoft.com/?kbid=170619>

“Windows NT 4.0 ServiceProvider Priority Values Not Applied”  
<http://support.microsoft.com/?kbid=171567>

“Windows TCP/IP Registry Entries”  
<http://support.microsoft.com/?kbid=158474>

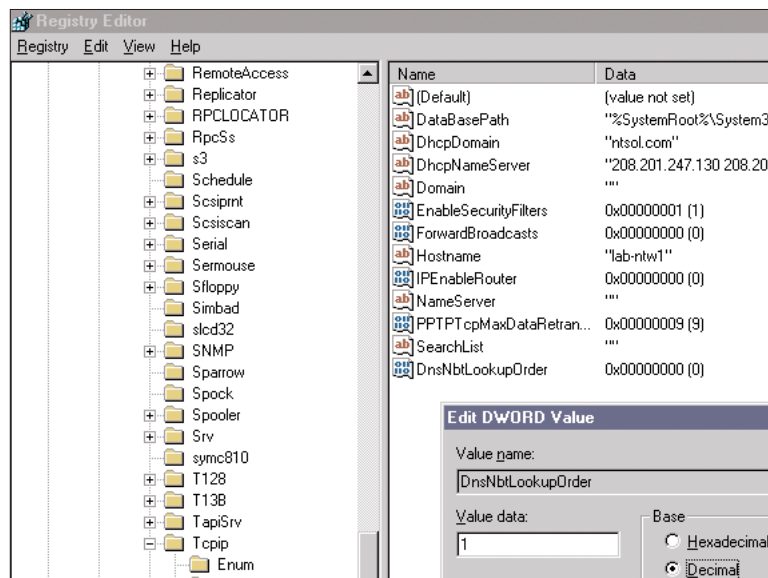
These values explain the client's preference for DNS (i.e., DnsPriority = 2000) over WINS (i.e., NetbtPriority = 2001). As you might expect, the article suggests that you simply reorder the values so that DNS has a higher value (and thus a lower priority) than NetBT and WINS. This suggestion seemed straightforward, so I implemented the changes on my Win95 OEM Service Release 2 (OSR2) test system and rebooted. To my dismay, I detected no difference in the client's name-resolution order: My network monitor showed that the client was still querying DNS first. Another Microsoft Web site search yielded a different article, “Windows 95 ServiceProvider Priority Values Not Applied” (<http://support.microsoft.com/?kbid=170619>). It essentially says, “Whoops, sorry, we messed up in the first article.” Apparently, the previous article's suggested changes don't work because earlier Winsock versions (i.e., versions earlier than wsock32.dll 4.00.1112) contain a bug that causes the system to read the registry data from the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\MSTCP\ServiceProvider registry key. The article suggests that to correct the problem, you can manually mirror the values at

the original, correct location to the incorrectly queried location. But I was shocked to discover that after I made these changes, my Win95 OSR2 client still queried DNS first. Even after I upgraded the client to Win98, I couldn't make any difference in my client's name-resolution order.

Just when I had all but given up on these registry values, I installed Win98SE, and my client suddenly began querying WINS before DNS. Puzzled over these inconsistencies, I returned to the Microsoft Web site. I found some articles that implied that the registry values only work with Winsock 1.x and some articles that implied otherwise. I tried everything I could think of, but I never successfully reversed the order on Win95 OSR2 with Winsock 2.x. I might have been more successful using the Win95 retail version, and I don't know whether these problems were the result of buggy code, inaccurate Microsoft documentation, or a combination of the two. However, the experience left a bitter taste in my mouth about the reliability of some Microsoft articles.

Fresh from my Win95 defeat, I decided to tackle the recommended solution for NT clients. I had a much more pleasant experience than with Win95. The NT modification takes the form of a registry `DnsNbtLookupOrder` subkey (naming is case sensitive) of type `REG_DWORD`. This subkey doesn't exist by default, so you must create it in the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters` registry key. The subkey can have a value of 0 or 1. The default value (i.e., 0) instructs the client to use DNS before it uses NetBT methods; a value of 1 instructs the client to first use NetBT methods. As Figure 3 shows, I added the subkey and set its value to 1, and the client system began querying WINS before DNS.

**Figure 3:**  
*Creating and modifying the `DnsNbtLookupOrder` registry subkey*



What's the downside to using this method to change a client's name-resolution order? Potentially, this solution can introduce a minor (i.e., several-second) delay when clients query names that aren't on the local LAN or that aren't resolvable using WINS. For example, I noticed that browsing to an Internet-based Web site on my newly modified client took a few seconds longer than it had taken previously (although the WINS timeouts were much faster than the DNS timeouts I'd experienced earlier). Whether this trade-off is acceptable will probably depend on how much time the user in question spends on browsing the Web and engaging in other Internet-related activities vs. how much time the user spends connected to the company network resolving internal server names. Because most users spend a significant amount of time accessing the Internet, the best solution in most cases will probably be to upgrade to NT SP4 or later (or to Win98SE on Win9x systems). Another alternative is to upgrade to Win2K Professional, which also has quick DNS server timeouts.

In addition, not all name queries are created equal. A Windows client or application can generate two kinds of name queries: NetBIOS name queries and host name queries. Applications that send NetBIOS name queries bypass DNS and use the NetBT methods (which usually begin with WINS). Other applications and utilities make generic name queries, which the client handles as host name queries and hands off first to DNS. (You might be surprised to discover that even Microsoft Outlook uses generic host name queries to resolve the name of its Exchange servers. Because Outlook's server field can accept Internet FQDNs the application isn't limited to using only NetBIOS names.) Thus, these applications will suffer when DNS-timeout-related problems occur on a client PC. Most non-NetBIOS-centric commands and utilities, such as the Ping utility, issue host name queries as well. However, some applications and utilities bypass DNS and pass queries directly to NetBT. For example, when you type the Universal Naming Convention (UNC) phrase

```
\\SERVER1
```

in the Start, Run dialog box to bring up a list of shared resources on a server named SERVER1, an h-node client will process the request as a NetBIOS name query, as Figure 4 shows. Other UNC-style name references, such as those you use in drive mapping for logon scripts, also exhibit this behavior. However, on my network, I noticed more generated host name queries than NetBIOS name queries.

## Knowledge Is Power

Armed with an understanding of Windows clients' name-resolution behaviors on IP networks and with the techniques I've discussed for changing these behaviors, you're in a better position to control and optimize your network's performance. Now that you're aware of these lesser-known behaviors, you can use that knowledge to navigate the sometimes-troubled waters of name resolution.

**Figure 4:**  
*Generating a NetBIOS name-resolution request*

The screenshot shows the NetXRay application window. The main pane displays a list of network packets with the following columns: No., Status, Source Address, Dest Address, Layer, and Summary. Packet 2 is highlighted, showing a NetBIOS-NS request from 208.201.247.194 to 208.201.247.230. Below the list, the packet details for packet 2 are expanded, showing a NetBIOS Name Service packet with a transaction identifier of 30 and various flags.

No.	Status	Source Address	Dest Address	Layer	Summary
1	Ok	208.201.247.230	208.201.247.194	NetBios-NS	C=Name Query
2	Ok	208.201.247.194	208.201.247.230	NetBios-NS	R=Pos-Name Q
3	Ok	208.201.247.230	208.201.247.196	TCP	1085->NETBIO
4	Ok	208.201.247.196	208.201.247.230	TCP	NETBIOS Sess
5	Ok	208.201.247.230	208.201.247.196	TCP	1085->NETBIO
6	Ok	208.201.247.230	208.201.247.196	NetBios-SS	Session Requ
7	Ok	208.201.247.196	208.201.247.230	NetBios-SS	Positive Ses
8	Ok	208.201.247.230	208.201.247.196	SMB	C=Negotiate,
9	Ok	208.201.247.196	208.201.247.230	SMB	R=Negotiate,
10	Ok	208.201.247.230	208.201.247.196	SMB	C=Session_Se
11	Ok	208.201.247.196	208.201.247.230	SMB	R=Session_Se
12	Ok	208.201.247.230	208.201.247.196	SMB	C=Transactio
13	Ok	208.201.247.196	208.201.247.230	SMB	R=Transactio
14	Ok	208.201.247.230	208.201.247.196	SMB	C=Transactio
15	Ok	208.201.247.196	208.201.247.230	SMB	R=Transactio
16	Ok	208.201.247.230	208.201.247.196	SMB	C=Transactio
17	Ok	208.201.247.196	208.201.247.230	SMB	R=Transactio

Packet Details (Packet 2):

- Checksum: 0x13A9
- NETBIOS Name Service
- HEADER SECTION:
- Transaction Identifier: 30
- Flags: Req QUERY Non-Auth Recv Desr Recv Not Ava Unicast ECode=No

## Chapter 6

# Troubleshooting DNS-Related AD Logon Problems

—by *Mark Minasi*

I get many reader questions that go something like this: “I set up a test computer called DC1 at 192.168.1.4 to play with Active Directory (AD). This computer runs Windows 2000 Server and acts as both the DNS server and the first AD domain controller (DC) on my test domain, which I call acme.com. Everything works fine with that DC—I can create users and run all the AD administrative tools. Then, I put Win2K Server on another computer named DC2 at 192.168.1.5 and run Dcpromo to make that computer the second DC in the domain. But Dcpromo says that it’s unable to contact the domain and can’t make DC2 into a DC for acme.com. Yet I can ping DC1 from DC2 and vice versa, and the DCs are on the same subnet, so why can’t they see each other?”

What causes this problem? To see how things go wrong, let’s reconstruct what happens.

## Finding a DNS Server

When you run Dcpromo and instruct it to make your computer into an additional DC, Dcpromo displays a Network Credentials panel that requests the name, domain, and password of an account that has the authority to add a DC to the existing domain (e.g., acme.com). Dcpromo then looks for an acme.com DC and uses the name and password to try to log on. But first, Dcpromo must be able to find a DC.

Remember that DNS acts as the central naming service for an AD domain. An essential DNS function in a Win2K network is to help computers find DCs. To use DNS, your system needs to have two pieces of software: the server software, which runs on the DNS server, and the DNS client software, which runs on your workstation. Your workstation uses the DNS client software to resolve names. But the client can’t help unless it can find a DNS server. To find a DC that will authenticate you, Dcpromo says to the DNS client software, “Find a DNS server for acme.com, and ask it for the names and IP addresses of the DCs in the domain.” The client does so by querying the DNS server for an SRV record.

To determine which DNS server your system queries, you can open a command line and type

```
ipconfig /all
```

Within the output, you’ll see a list of the IP addresses for all the domain’s DNS servers. When the DNS client software needs to resolve a DNS query, the client tries to contact the first machine on the list. If a DNS server is at that location, the client will address all DNS queries to that machine. The client doesn’t query another DNS server from the list unless the client gets no DNS server response from the preferred server.

When a Win2K system needs to find an AD DC to log the system on, the system prefers to use a local DC. So, a Win2K system often does multiple DNS queries when looking for a DC: first, “Tell me about the local DCs for acme.com,” then if that query fails, “Tell me about all the DCs for acme.com.”

## Troubleshooting AD Authentication

A powerful utility for troubleshooting AD authentication problems is Nslookup, which lets you mimic the behavior of a Win2K system that’s trying to log on to an AD domain. On a command line, type

```
nslookup
```

The response gives the default server and its IP address and tells you that Nslookup did two things: It successfully contacted a DNS server, and it asked the server to reverse-resolve the server’s IP address into a DNS name. Reverse resolution is unnecessary for AD logons, but if a functional DNS server doesn’t reverse-resolve, Nslookup returns a message that makes you think something is seriously wrong:

```
DNS request timed out. timeout was 2 seconds.
*** Can't find server name for address 200.200.10.10: Timed out
*** Default servers are not available
Default Server: UnKnown
Address: 200.200.10.20
```

However, if you type an Internet address (e.g., www.win2000mag.com) at the Nslookup prompt, the response reveals that Nslookup easily resolves that name:

```
www.win2000mag.com
Server: ns1.yourisp.com
Address: 200.200.10.10

Name: www.win2000mag.com
Address: 63.88.172.66
```

Nslookup also complains if the DNS server that your DNS client is supposed to use is dead or has a problem:

```
*** Can't find server name for address 200.200.10.10: No response from server
*** Default servers are not available
Default Server: UnKnown
Address: 200.200.10.10
```

Do you see a significant difference between this response and the one that indicates a failed reverse resolution? No? Neither do I. So, how can you determine whether the server has a real problem or Nslookup is complaining about reverse resolution? Your best bet is to simply try to resolve an Internet address. If the problem exists at the server, the response will instead resemble

```
Server: UnKnown
Address: 200.200.10.20
```

```
*** UnKnown can't find www.win2000mag.com: No response from server
```

Now let's ask Nslookup to simulate using a local DC to log on. You'll need to know the name of your domain and your AD site. You'll always have at least one site: When you create the first DC in a forest, Dcpromo creates a site called, by default, Default-First-Site-Name. Type the Nslookup command on a command line. At the prompt, enter two commands, using the syntax

```
set type=srv
_kerberos._tcp.<SiteName>._sites.dc._msdcs.<DomainName>
```

Be sure to include the underscores, and type the second command on one line without spaces. For example, for a site named HQ in the acme.com domain, you'd type

```
set type=srv
_kerberos._tcp.hq._sites.dc._msdcs.acme.com
```

If you type the commands correctly, you might get a response like the one Figure 1 shows (don't worry if you don't—that's why we're troubleshooting). But if those commands don't work for you, then Dcpromo won't be able to log on either. Dcpromo would then say to DNS, "Tell me about all the DCs in the world for acme.com." You can also use Nslookup to simulate that query:

```
_kerberos._tcp.dc._msdcs.acme.com
```

But if your first query failed, this one likely will too, leading to a message resembling *ns1.yourisp.com can't find kerberos.\_tcp.dc.\_msdcs.acme.com: Non-existent domain*.

The reason for this problem in most test networks is that your AD domain (e.g., acme.com) has the same name as a registered Internet domain, and that name conflict poses problems. To see why, let's roll back the clock to when you created the first DC in your test forest.

### Figure 1:

*Sample response from logon commands*

```
Server: ns1.yourisp.com
Address: 200.200.10.10

_kerberos._tcp.headquarters._sites.dc._msdcs
.acme.com SRV service location:
    priority = 0
    weight = 100
    port = 88
    svr hostname = dc1.acme.com
dc1.acme.com internet address = 192.168.1.4
```

## The Origin of the Problem

When you use Dcpromo to create a new forest with a first domain named acme.com, Dcpromo needs to write several records into the writable copy of the acme.com zone, which lives on the domain's primary DNS server. So, Dcpromo queries the local DNS server for the address of the primary DNS server for acme.com. Because you're just playing around from home, your Win2K server likely connects to the Internet through DSL or a cable modem and thus points to some DNS server on the Internet. That DNS server happens to know about a registered Internet domain

named `acme.com` and responds that the primary `acme.com` server is some UNIX box on the Internet.

`Dcpromo` then says to that distant server, “Hi. I’m about to write a bunch of new SRV records to you with dynamic DNS (DDNS). That’s all right, isn’t it?” The UNIX server responds, “No! I don’t know you, and I’m not about to let you write records into my zone.” That response, of course, leaves `Dcpromo` in a quandary.

Instead of telling you that it has found the `acme.com` DNS server but the server doesn’t accept updates, `Dcpromo` fibs by reporting that it couldn’t find the DNS server for `acme.com` and offers an alternative: “Would you like me to configure a DNS server for this domain?” You say “Yes,” grateful that `Dcpromo` is such a can-do kind of program. So, `Dcpromo` sets up the soon-to-be DC as a DNS server, creates an `acme.com` zone on that server, uses that zone to set up `acme.com`, then reboots. That’s when the trouble starts.

## A Lack of Communication

When setting up a local DNS server, `Dcpromo` creates a zone with the same name as the domain—`acme.com`, in my example. Thus, when you create a new AD domain, your computer serves two functions: It’s both the first DC and the DNS server for the domain. But because `Dcpromo` never tells the DC software that the DNS software exists on the same computer, the DC still can’t find a DNS server that will accept dynamic updates for `acme.com`.

In other words, `Dcpromo` sets up the DNS server service and the `acme.com` zone but doesn’t tell the computer’s TCP/IP stack to refer to itself when looking for a DNS server. The DNS server is running, but no computer—not even the computer that the DNS server is running on—knows that it needs to refer to that DNS server first when looking for a domain. So, `Dcpromo` completes the basic domain setup, then asks for and receives permission to reboot the computer.

As it reboots, the computer sets up a TCP/IP stack just as you had set up the stack before you made the computer a DC. At that time, you used DHCP either to configure the stack or set it up statically. If you used DHCP, the DHCP server is either your ISP’s DHCP server or your corporate DHCP server. Regardless, the DHCP server is unlikely to tell your computer to use itself as a preferred DNS server. If you originally configured your TCP/IP stack statically, did you tell the computer to use itself as a preferred DNS server? Probably not. (But if you did, congratulations—good work!)

As a result, when the computer boots, its DHCP or static configuration tells the computer to use some DNS server other than itself—probably some server on the Internet. Then, the Netlogon service swings into action.

As on Windows NT 4.0, Netlogon on Win2K is an important service that runs only on DCs. Among Netlogon’s first duties are to locate the primary DNS server for its domain (e.g., `acme.com`), then to use DDNS to write the server identification (SRV) and host name (A) records for Netlogon into the domain’s zone. The Netlogon service on each DC periodically reintroduces itself to the domain’s DNS zone so that DNS knows about that DC. After the `acme.com` zone knows about a DC, workstations and other DCs in `acme.com` can find that DC.

But when the `acme.com` DC’s TCP/IP stack points to a DNS server other than itself, a query for the primary DNS server ends up on the public Internet’s DNS servers and returns the address of the DNS server for the registered `acme.com` domain. When Netlogon tries to write records to that UNIX DNS server, the server refuses the updates and Netlogon reports the problem in the

System log as event ID 5773: *The DNS server for this DC does not support dynamic DNS. Add the DNS records from the file '%SystemRoot%\System32\Config\netlogon.dns' to the DNS server serving the domain referenced in that file.*

If you can't convince a workstation to log on to a new AD domain or if Dcpromo refuses to run on a second machine that you intend to be another DC in your domain, check the System log for event ID 5773. If Dcpromo told you that it would configure DNS for you, event ID 5773 is proof that you need to point your DC's TCP/IP stack to itself.

For further proof, start up the Microsoft Management Console (MMC) DNS snap-in, double-click the icon for your server, open the Forward Lookup Zones folder, then double-click the folder for your domain. A folder that contains only two or three records and no other folders indicates that Netlogon couldn't find the DNS server that's running on the same computer as Netlogon. Configure your computer's TCP/IP settings to point to your computer as the preferred DNS server, restart Netlogon, then again check the folder for your domain; it will contain four folders full of find-a-DC information.

## Adding a Second DC

But you're still not out of the woods. Your first DC can now find itself, but a second DC won't be able to find the first one. After you put Win2K on a second machine and join that machine to acme.com, you need to configure that machine's TCP/IP stack to look to the first machine as its preferred DNS server. You can do the same for any systems that you want to make into additional DCs.

If you want your network to grow, you can make a few more Win2K servers into DNS servers for acme.com. Because Dcpromo automatically makes the acme.com zone into an AD-integrated zone, you can use only DCs as acme.com DNS servers. If you don't want an AD-integrated zone—perhaps because you want to use non-DCs as DNS servers in the domain—you can convert the AD-integrated zone into a standard primary zone with a few clicks. Simply open the DNS snap-in, double-click the DNS server's icon, then open the Forward Lookup Zones folder. Inside, you'll see a folder labeled with your domain name. Right-click that folder, choose Properties, then click the General tab. Click Change and select *Standard primary* instead of *Active Directory-integrated*.

## Building It Right

By now, our DNS infrastructure is in pretty good shape. Let's look at how to expand it to larger networks. But instead of letting Dcpromo discover that a problem exists with DNS, let's make sure that DNS works from the start.

Suppose you actually work for acme.com and are going to set up AD for the company. Would that be a simpler scenario? You'd just replace the existing DNS servers with servers that accept dynamic updates, right? Probably not. AD stores information in a DNS zone that you might not be comfortable advertising on the public Internet. So even if you work for acme.com, you'd still want to fake out AD by letting it use an internal-only DNS, thereby causing AD to leave the publicly visible DNS untouched. Such a setup is called split-brain DNS and is a good idea for any enterprise, security-wise. Suppose Acme's ISP hosts the publicly visible acme.com zone. Here's how you'd set up an internal-only DNS for acme.com's AD.

First, set up at least one more DNS server on the acme.com intranet. On the first server that you set up, make the acme.com DNS zone a standard primary zone and enable dynamic updates. On any subsequent servers, set up acme.com as a standard secondary zone. (You can even use

NT 4.0 servers as secondary DNS servers in AD, provided that those NT servers run Service Pack 6a—SP6a.) Configure the TCP/IP stacks on each workstation and server in your intranet to use one of your internal DNS servers as the preferred DNS server and another DNS server as the alternative DNS server.

Now, when you run Dcpromo, it will find the acme.com DNS server that you want it to find—your internal DNS server—and will set up AD correctly. Workstations and servers will be able to find that DC to log them on, and when you run Dcpromo on other systems to create additional DCs, Dcpromo will run fine. After you set up AD, you can shift your zone to AD-integrated.

Now that you have two sets of books DNS-wise, you have only one more thing to do. If someone inside Acme tries to find `www.acme.com`, that user will always find the zones on the intranet server and won't be able to find the public zone. So, be sure to manually copy the records in the public zone to the intranet zones. If you don't, you'll discover that the only people who can't see Acme's Web site are the people inside Acme, which makes for a strange situation.

## Chapter 7

# Scavenging Stale DNS Records

—by *Mark Minasi*

Because dynamic DNS (DDNS) and WINS are in many ways similar (both name-resolution systems maintain a database that lets the system find a machine's IP address, given the machine's name), I have a subliminal tendency to assume that all I need to do to plan for DDNS is dust off my Windows NT 4.0 knowledge about WINS servers and replace all mention of WINS with DDNS and all references to NT with Windows 2000. Although that bit of cognitive laziness isn't entirely incorrect, neither is it completely accurate. In Chapter 3, I highlighted some of the differences between how WINS and DDNS add new records to their machine-name-to-IP-address databases. In this chapter, I look at how DDNS lets you get rid of old records—stale records is the Win2K term—in its database.

Before you continue, however, you should know that DDNS offers two ways to store that database. The more traditional—and compatible with non-Win2K DNS servers—method is zone files, and the newer method is known as Active Directory (AD)-integrated zones. I limit this chapter to discussing traditional zone files.

## A Time to Live

By default, when a machine registers its name and IP address in a DDNS database (i.e., the set of zone files that the DNS server stores), DDNS remembers that information forever. DDNS stores the information as a host record, also known as an *A record*. If a machine named mypc with an IP address of 100.100.20.17 is in a DDNS domain named acme.com, mypc's name record in the acme.com zone file would look like

```
mypc 1200 IN A 100.100.20.17
```

Translated, this record says that the machine named mypc.acme.com has an IP address of 100.100.20.17. The term IN stands for Internet and represents the class of the DNS record, and the A means that the record is a simple name-to-address record (DNS has many types of records).

The number 1200 in the name record is the Time to Live (TTL) value. The TTL value tells other DNS servers how long they can expect the address to remain valid. For example, mypc's TTL value of 1200 tells another DNS server that it can expect mypc.acme.com to still have the same address 1200 seconds (i.e., 20 minutes) from the time the DNS server reads the record. So, the other server doesn't need to recheck the address if the server needs the same information again within 20 minutes. However, the server shouldn't assume that the address information will remain correct beyond that time.

The DDNS server doesn't assign the TTL value; rather, the client computer (e.g., mypc) specifies the TTL value when the client computer registers its name with the DDNS server. Note that the TTL value doesn't tell the DDNS server to delete the A record after the TTL expires—the TTL is

simply an instruction to other DNS servers advising them not to cache the A record for longer than the specified time.

## And a Time to Scavenge

Let's assume that we boot mypc, which registers its A record with the primary DDNS server for the acme.com zone. Then, we shut down mypc and never turn it on again. A year later, that A record for mypc will still exist in the acme.com zone file because, by default, Win2K DDNS doesn't eliminate old records. But you can configure scavenging to change that behavior and direct Win2K's DDNS server to eliminate stale records of all types—not only A records.

To activate scavenging, you need to turn it on in several places. First, open the Microsoft Management Console (MMC) DNS snap-in and right-click the icon in the left-hand pane that represents your DNS server. Choose Set Aging/Scavenging for all zones, and select the Scavenge stale resource records check box in the Server Aging/Scavenging Properties dialog box. You'll see No-refresh interval and Refresh interval controls. I'll discuss these intervals later; for now, simply use the defaults. Click OK to clear the dialog box, then click OK again in the Server Aging/Scavenging Confirmation dialog box. Repeat these steps for each zone.

Assuming your DNS server is the primary DNS server for the acme.com domain, double-click the icon that represents the DNS server to see the Forward Lookup Zones folder. In that folder, you'll see another folder that represents the acme.com domain. Right-click the domain folder, choose Properties, and click Aging on the General tab. Again, select the Scavenge stale resource records check box and click OK, then click OK again.

Finally, right-click the icon that represents your DNS server, choose Properties, and click the Advanced tab on the resulting properties page. Select the Enable automatic scavenging of stale records check box, and you're finished.

## Scavenging Periods and Refresh Intervals

Now, let's go back and look at what we did. If you visited all the dialog boxes that I described above, you might have noticed three time intervals: Scavenging period, No-refresh interval, and Refresh interval. (The Server Aging/Scavenging Properties dialog box contained only the no-refresh interval and the refresh interval, but that dialog box seems to apply only to new zones. The No-refresh interval and Refresh interval controls that appear on the dialog you raised by clicking Aging on the zone's Properties page seem to be the intervals that count.) Let's examine how these three intervals interact.

The scavenging period simply tells your DDNS server how often to look in its zone files for records that are old or that are for computers that haven't reregistered in a long time. You might have records that have been stale for years, but the DNS server won't delete them until the scavenging feature runs.

As I mentioned, Win2K DDNS lets you store a DNS zone's information in AD. Because AD replication can consume a lot of bandwidth, Microsoft was concerned that DDNS clients that frequently reregistered with DDNS would set off a lot of AD replication. Consequently, Microsoft incorporated the no-refresh interval into Win2K DDNS. Suppose you set the no-refresh interval to 5 days. If mypc registers its name and address with a DDNS server, that DDNS server will ignore mypc's attempts to reregister within 5 days. For example, if mypc registers at 9:00 A.M. Monday, it won't be able to reregister until 9:00 A.M. Saturday.

You can think of the refresh interval as defining the end of the no-scavenge period. Recall that scavenging runs at a regular interval, which you can specify. When the scavenging routine runs, it examines each record in a DNS zone and determines whether the record is stale. If it is, the scavenging feature deletes the record.

But what criterion does scavenging use to determine at what age a record becomes stale? When a record is older than the sum of the no-refresh interval and the refresh interval, the scavenging feature considers the record stale and deletes it. So, when you set No-refresh interval to 3 days and Refresh interval to 5 days, scavenging will delete records that are more than 8 days old.

## Setting Intervals

You can set the scavenging period to any value you want. However, keep in mind that scavenging burns up CPU time. Thus, on one hand, the smaller the value (i.e., the more frequently you run scavenging), the more scavenging will slow performance. On the other hand, a larger scavenging interval lets stale records remain in the database longer. I'd err on the side of a larger value—say, 2 weeks or so.

On a network that's fairly stable, I suggest setting the no-refresh interval to 1 week or 2 weeks to minimize the morning rush to reregister with the domain's primary DDNS server. Set the refresh interval so that the sum of the no-refresh and refresh intervals (which I think of as the "mandatory retirement age") gives your system a chance to reregister.

Remember that the client, not the server, initiates reregistration, and ask yourself how often your system will try to reregister. PCs reregister with DNS whenever their DHCP leases are renewed; when the PC's static IP address changes; when you reboot the system; every 24 hours, if you leave the PC running for that long; or when you run the `Ipconfig /registerdns` command. So, if you reboot your PC every day, your PC will attempt to reregister every day. If your system is always running, it will reregister once a day. If you were to leave your system running and wait for a DHCP renewal, reregistration would occur in 4 days (half the default of 8 days for a DHCP lease under Win2K). But wait; if you left the system up all the time, the system would reregister every 24 hours without waiting for the DHCP lease to expire. Thus, the longest time that mypc would ever wait to try to reregister would be 1 day.

Of course, you don't want the intervals to be too short. For example, if you set mypc's no-refresh interval and refresh interval both to 1 hour, DDNS records would expire after 2 hours, leaving 22 hours before mypc tried to reregister. Obviously, that mandatory retirement age wouldn't work. The bottom line is this: No matter what value you choose for the no-refresh interval, make sure that the sum of the refresh interval and the no-refresh interval exceeds 24 hours.

Do all these ins and outs make DDNS seem ugly? It isn't, by WINS standards: WINS has more complications for you to worry about, including four intervals and tombstoning. Win2K DDNS can also scavenge reverse lookup zones, causing stale PTR records to expire. Overall, I'd say that DDNS is a good thing compared with WINS.

## Chapter 8

# Integrating UNIX DNS with Windows 2000

—by *Tao Zhou*

DNS is a standard name-resolution mechanism that an IP host uses to name, advertise, and locate IP hosts and services on the Internet and over an intranet. To fully integrate Windows 2000 into the Internet, Microsoft used standard DNS rather than WINS as Win2K's native naming service. In addition, Microsoft has implemented dynamic DNS (DDNS) to let Win2K systems dynamically register their names and IP addresses in a DDNS server. Dynamic updates of machine information eliminate the administrative burden of maintaining a static database in a traditional DNS server. Win2K domain controllers (DCs) also dynamically register their SRV records in DDNS servers. Clients in a Win2K network look up SRV records in the DDNS server to locate the network's Active Directory (AD) and its services (e.g., logon service).

Internet Engineering Task Force (IETF) Request for Comments (RFC) 2052 documents SRV records, and RFC 2136 documents DDNS updates. SRV record and DDNS updates aren't new to the family of DNS standards—IETF published RFC 2052 in October 1996 and RFC 2136 in April 1997. However, Win2K's use of these features challenges companies that have long run DNS on UNIX machines to map host names and IP addresses. Most of these companies haven't upgraded to the most recent version of Internet Software Consortium's (ISC's) Berkeley Internet Name Domain (BIND) 8.2.2 or to a new DNS version from their UNIX and DNS vendors that supports both SRV record and dynamic updates. If this scenario describes your company and you're planning a Win2K deployment, you face an immediate integration question: Should you migrate to Win2K DNS or continue to use UNIX DNS? You have three options: migrate to Win2K DNS, create an environment in which UNIX and Win2K DNS coexist, or use only UNIX DNS.

Before I delve into these options, I describe how Win2K uses SRV records to help you better understand the benefits and drawbacks of each option and dispel any confusion about SRV records' use of underscores. Understanding Win2K's dynamic updates and how to examine your UNIX DNS server's dynamic update support will also help you decide which option is best for your company.

## SRV Records in Win2K

Windows NT clients don't need to specify an NT DC when they log on to an NT domain. NT WINS provides a DC to the client for logon. This setup lets clients access an NT domain as long as one DC is available, even if all other DCs have failed. Win2K inherited this functionality and lets clients log on to an AD domain without specifying a DC. However, to provide this service to its clients, Win2K uses DNS rather than WINS. (Win2K provides WINS for backward compatibility.) When a Win2K client system logs on, it queries the DNS server for the DCs of the logon domain. Win2K uses SRV records to locate the logon service, then sends the client the DCs' names. The client uses an available DC to log on to the AD domain.

SRV records provide a general mechanism for advertising and locating Internet services. For example, suppose `acme.com` is your company's Web server. Pointing an SRV record of `_http._tcp`

.acme.com to webdev.acme.com lets you enable an SRV record-capable browser to use <http://acme.com> instead of <http://webdev.acme.com> to access your Web server. The SRV record will instruct the browser to use the webdev.acme.com server. This functionality means you don't need a Canonical Name (CNAME) of webdev.acme.com in the acme.com DNS domain, so you can easily move the Web service from one server to another without a hassle. SRV records also enable load balancing and fault tolerance when you associate two or more servers with the same service.

Let's take a look at how Win2K uses SRV records. If the AD domain (i.e., DNS domain) acme.com contains two DCs, w2kdc1 and w2kdc2, Win2K uses two SRV records for the Lightweight Directory Access Protocol (LDAP) service in the acme.com domain. The following SRV records advertise and locate the two LDAP-enabled DCs in the acme.com domain: `_ldap._tcp.acme.com. 600 IN SRV 0 100 389 w2kdc1.acme.com.` and `_ldap._tcp.acme.com. 600 IN SRV 0 100 389 w2kdc2.acme.com.` When a client queries `_ldap._tcp.acme.com` looking for a DC, acme.com's DNS server will send the client these two records. The client picks either w2kdc1.acme.com or w2kdc2.acme.com as the DC and uses the LDAP protocol at TCP port 389 to communicate with the DC.

The client chooses a DC based on the parameters in the SRV records and the DCs' availability. A DNS server and Win2K system cache receive DNS records. The number that follows the protocol and domain name (i.e., 600 in my example) is the caching Time to Live (TTL) value. For example, Win2K can cache the records in my example for 600 seconds (i.e., 10 minutes). The number following SRV (i.e., 0 in my example records) shows the record's priority. The smaller the number, the higher the priority. The number that follows the priority (i.e., 100 in my example records) shows the record's weight. The larger the number, the heavier the weight. If multiple same-service records have the same priority but different weights, the client prefers a record with a heavier weight. If multiple same-service records have the same priority and weight, the client communicates with the DCs in a round-robin fashion. If the client can't connect to a chosen DC, the client goes to the next preferred DC. In this way, SRV records provide load balancing and fault tolerance.

Win2K also uses SRV records to advertise and locate DCs in a specific domain, DCs in a specific site, a PDC in a specific domain (for a mixed-mode Win2K and NT installation), Global Catalog (GC) servers in a specific domain, GC servers in a specific site, Kerberos Key Distribution Centers (KDCs) in a specific domain, and Kerberos KDCs in a specific site. For a list of the SRV records that DCs register in DNS, see the "DNS Record Registration and Resolver Requirements" section of the Microsoft white paper "Windows 2000 DNS" at <http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windows2000serv/plan/w2kdns2.asp>.

Some users complain about Microsoft's use of underscores ( `_` ) in Win2K's SRV records because some UNIX DNS implementations don't accept underscores. Microsoft isn't to blame for the use of underscores. In January 1999, IETF updated RFC 2052 in an Internet Draft. IETF subsequently replaced RFC 2052 with RFC 2782 (<http://www.ietf.org/rfc/rfc2782.txt?number=2782>). Microsoft adopted the updated version for SRV records in Win2K because the new version replaced the old RFC. In the new version, the IETF appends an underscore to SRV records' service identifier and protocol name to differentiate the service identifier and protocol name from other DNS labels. The updated SRV record format is `_service._protocol.domainname. TTL Class SRV RR Priority Weight Port Host`. Many UNIX DNS implementations, such as BIND 4.9.4 and later, check to ensure that host names conform to RFC 952, a host-naming standard that doesn't allow underscores in a host name. However, the first part of the updated SRV record that contains underscores (formally called the SRV record's owner) isn't a host name, so the underscores aren't a problem. In addition, a DNS

implementation that conforms to RFC 952 doesn't check host names against an SRV record's owner, and DNS domain names can contain underscores. In my experience, underscores in SRV records haven't posed a problem to BIND 8.2.2. If your version of UNIX DNS doesn't accept underscores in SRV records, change the check-names option on the UNIX DNS server from the default *fail* setting to the *warn* setting or the *ignore the underscores* setting.

## DDNS in Win2K

To install or promote a Win2K DC, the domain's authoritative DNS server must support dynamic updates to correctly register the DC's AD and Netlogon services in DNS. The netlogon.dns file in the C:\winnt\system32\config directory contains about 20 SRV record, A, and CNAME records that the DC needs to register on a DNS server. Win2K lets you continue a DC installation if the OS discovers that the DNS server doesn't support dynamic updates, but you must manually enter the records into the domain's zone file on the DNS server after the installation. If you don't want to use a DDNS server, manually registering these records might work in a Win2K network that contains a very small number of DCs. However, midsized and large networks should seriously consider using Win2K's DDNS or a third-party DDNS system to reduce the administrative burden of entering records and eliminate the risk of introducing errors.

Win2K client systems also use dynamic updates. When it boots, a Win2K client that has a static IP address will try to register its A and PTR records in its forward and reverse lookup zones on the DDNS server. A workstation that uses DHCP to lease a dynamic IP address will try to register its A record in the DDNS server when the workstation receives the IP address. The DHCP server will try to register the workstation's PTR record in the DDNS server for the workstation. To be backward compatible, the Win2K DHCP server will register both A and PTR records of NT and Windows 9x DHCP clients in the DDNS server. To fully use Win2K's dynamic updates, you must have a DDNS system in place when you deploy Win2K.

UNIX DNS can usually accept dynamic update requests of A, PTR, and some other kinds of records. However, UNIX DNS might not support dynamic updates of SRV records. For example, BIND 8.2.1 doesn't support dynamic updates of SRV records, and early versions of BIND 8 are unlikely to support dynamic updates of SRV records. To support Win2K, your UNIX DNS implementation must use BIND 8.2.2 or BIND 9.

To discover whether your UNIX DNS implementation supports dynamic updates of SRV records, you can use the command-line Nsupdate utility that comes with BIND to make a dynamic update of an SRV record. For example, create a test domain (e.g., acme.com) and its zone file in your DNS server. Turn on the dynamic update option of the zone, and let any host dynamically update the zone. Use the Nsupdate utility to manually generate a dynamic update of the SRV record for the Web service in acme.com:

```
$ nsupdate
> update add webdev.acme.com. 86400 IN A 192.168.1.10
> update add _http._tcp.acme.com. 3600 IN SRV 0 1 80 webdev.acme.com.
>
```

The first command starts the Nsupdate tool, the second command adds the Web server's A record, the third command adds the Web service SRV record to the acme.com domain, and the last (blank) line instructs Nsupdate to send the two dynamic updates you entered to the DNS server. After you run this

command, if you can use Nslookup or look in acme.com's zone file to find the two records in the memory of the DNS server, your DNS server supports dynamic updates of SRV records. BIND temporarily saves received dynamic updates in a zone log file, then modifies the zone file accordingly.

## Option 1: Migration

After you check your UNIX DNS implementation for SRV record and dynamic update support, you'll likely discover that your implementation doesn't provide this support. An easy way to support the deployment of Win2K in your company is to migrate your current DNS service to Win2K DNS. Win2K DNS supports SRV records, dynamic updates, and AD integration.

The migration procedure is straightforward. You install a standalone Win2K server before you install a DC and AD. Install Win2K's DNS service on the standalone server from the Networking Services dialog box. Copy the forward and reverse lookup zone files from your UNIX DNS server to the C:\winnt\system32\dns directory on the new Win2K DNS server.

You can use FTP to transfer the files from UNIX to Win2K. If you're not sure where the old zone files are in the UNIX machine, the directory statement in the named.boot or named.conf file in the /etc directory tells you which directory holds the zone files. UNIX DNS administrators often use a filename that starts with db to represent a zone file (e.g., db.acme). By default, Win2K uses a filename with a .dns extension to express a zone file (e.g., acme.com.dns), but you can use a different file-naming convention or use the old UNIX DNS zone filenames in Win2K DNS.

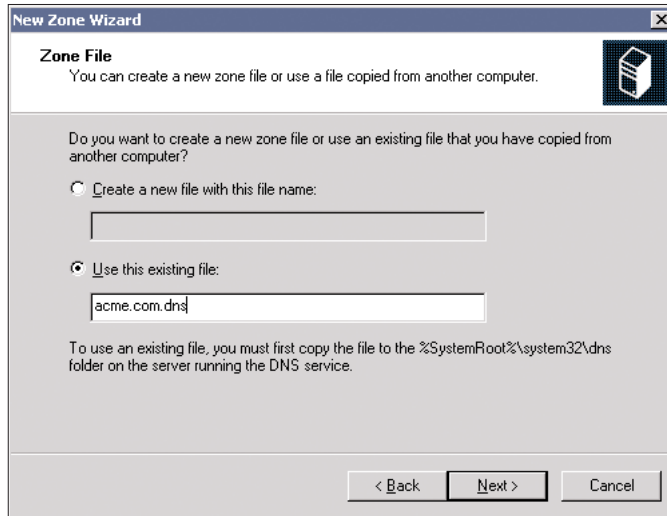
If your AD domain name is different from the UNIX DNS domain name, you must switch to the AD domain name for Win2K DNS. Change the old domain name in the copied zone files that are now on the Win2K DNS server to the new domain name.

You're now ready to create a new zone in the Win2K DNS administrative tool for each zone file that you copied from your UNIX DNS server. After you enter the domain name (e.g., acme.com), the New Zone Wizard prompts you to create a new zone file or use an existing zone file, as Figure 1 shows. Select the Use this existing file option, and enter the zone filename that you copied from UNIX to Win2K or the zone filename that you modified from the UNIX zone filename, such as db.acme or acme.com.dns. In the next screen, you select this DNS server to be the primary server that holds the master copy of the created zone. (BIND often calls this primary server the primary master server.) After you migrate the old zone file to the Win2K DNS server, you need to change the Allow dynamic updates setting to Yes in the General tab of the zone file's properties to enable dynamic updates for the zone. By default, Win2K sets this option to No. To enable load balancing and fault tolerance, you need to set up at least one secondary DNS server for each forward and reverse lookup zone.

With Win2K's DNS service in place, you can implement AD. If you promote a Win2K server that hosts the DNS server to a DC, you can integrate the DNS server into AD. All AD-integrated DNS servers are primary master servers, which store the zone information in AD and replicate one another through AD's multimaster replication mechanism. An AD-integrated DNS server also supports secure dynamic updates; this support lets you define which Win2K clients can perform a dynamic update.

If your intranet DNS servers are also your Internet name servers, you need to ask InterNIC (<http://www.internic.net>) to change your domain registration to your new server names and IP addresses. In addition, you need to ask the American Registry for Internet Numbers (ARIN—<http://www.arin.net>) to change your IN-ADDR registration to the new name servers for reverse lookup on the Internet.

**Figure 1:**  
*Creating a new Win2K DNS zone file*



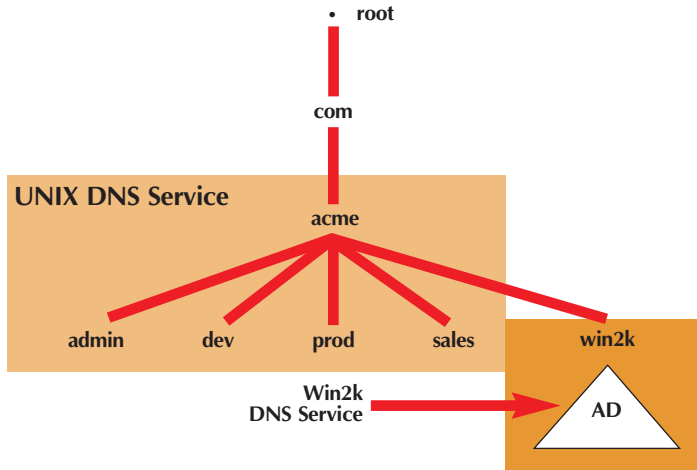
New Win2K clients and other clients can now use the new DNS servers. If you still have clients that use DHCP, you simply change old DNS server addresses to the new DNS server addresses in the DHCP server's configuration. The old client receives the new DNS server addresses the next time the client renews its IP address or the user restarts the computer.

## Option 2: Coexistence

Most companies running a UNIX DNS service have heterogeneous OS environments that include Linux, Novell NetWare, UNIX, and Windows systems. To minimize the interruption to non-Windows users when you deploy Win2K, you can create a DNS subdomain and dedicate it to AD. AD doesn't require a two-level domain name such as acme.com. You can set up your AD root domain as a subdomain (e.g., win2k.acme.com) in your company's UNIX DNS hierarchy. In Figure 2, win2k is a subdomain of the Acme Corporation's existing DNS system. The win2k subdomain uses the Win2K DNS service.

For the Win2K DNS service to function in the new subdomain, you need to create a separate DNS zone for the subdomain and delegate the subdomain to at least one Win2K DNS server. This DNS server becomes an authoritative name server and provides DNS service for the subdomain. For example, if you install two name servers, ns1.win2k.acme.com and ns2.win2k.acme.com, for the subdomain win2k.acme.com, you can delegate the subdomain to both name servers in the acme.com domain. The two name servers become authoritative name servers for the subdomain win2k.acme.com and provide fault tolerance and load balancing.

You complete the delegation by adding an NS record for the subdomain to the parent domain's zone file. For example, to delegate the subdomain win2k.acme.com to the name server ns1.win2k.acme.com in the acme.com domain, you enter the following BIND statement in the zone file of acme.com:

**Figure 2:***A Win2K subdomain in a UNIX DNS hierarchy*

```
win2k 86400 IN NS ns1.win2k.acme.com.
```

You also need to include an A record for the host ns1.win2k.acme.com in the acme.com zone file.

Next, you need to set up a reverse lookup zone for the Win2K subnet. If your Win2K systems will reside on a separate subnet, you can use Win2K DNS servers to perform reverse lookups of host names. If your Win2K and non-Windows computers reside on the same subnet, you need to use your UNIX DNS systems as name servers for reverse lookup zones or switch to Win2K DNS. If you don't plan to use Win2K DHCP to dynamically register PTR records, you can continue to use UNIX DNS for reverse lookup. If you plan to take advantage of Win2K DNS's dynamic PTR record updates and your UNIX DNS server doesn't support this functionality, you need to switch to a Win2K DNS server for reverse lookup.

Alternatively, Win2K and UNIX DNS servers can work together as primary and secondary name servers for the same zone. For example, in Figure 2, the Win2K DNS server that primarily serves the win2k subdomain can be a secondary name server for the acme.com zone. Using this setup, a Win2K user in win2k.acme.com can quickly resolve a UNIX host name from the local Win2K DNS server. If your UNIX DNS server supports SRV records, you can set up a UNIX name server as a secondary name server of the AD domain. However, if the UNIX DNS server doesn't support dynamic updates, don't make that server an authoritative name server for the zone. When a DNS client sends a dynamic update to the zone's authoritative name server, the authoritative server forwards the update to the zone's primary master server for a zone update. If an authoritative name server doesn't support dynamic updates, it can't understand or service an arriving dynamic update request. A UNIX DNS server can be a secondary server to a Win2K DNS server even if the Win2K DNS server is AD-integrated. However, an AD-integrated DNS server can't be a secondary name server.

### Option 3: Only UNIX DNS

Companies that have long run UNIX DNS servers might regard the network naming service as a part of UNIX administration and intend to use UNIX DNS throughout and after a Win2K deployment. If you're managing the UNIX DNS service for such a company, you probably need to upgrade your UNIX DNS servers to a version that supports SRV record and dynamic updates because Win2K and AD require this support. BIND is the most popular UNIX DNS implementation, so I'll focus on upgrading and using BIND to support Win2K. If you want to use an alternative UNIX DNS implementation, consult the vendor and request a new DNS version that supports SRV records and dynamic updates.

Developers at the University of California at Berkeley originally wrote BIND. ISC now maintains and develops BIND and has ported BIND to almost all UNIX platforms, including Linux. ISC has also ported BIND 8.2.2 to NT. BIND 9 changed the underlying BIND architecture to support fast-growing Internet zones (e.g., .com) and include additional new features. As I discussed previously, BIND 8.2.2 and later fully supports Win2K. (Because of security vulnerabilities, you should use the most recent BIND versions—currently 8.3.4 and 9.2.2.) BIND is open-source freeware that you can download from ISC's FTP server (<ftp://ftp.isc.org/isc/bind/src>).

After you unzip and tar (i.e., archive) the BIND distribution file, you'll find BIND source code located in the src directory under the directory you saved the BIND distribution file in. In the src/port directory, you'll find about 20 directories, such as solaris and winnt. Each OS-named directory contains a makefile file that lets you use a C compiler from your OS vendor or GNU's GCC compiler (<http://www.gnu.org/directory/gnu/gcc.html>) to easily compile BIND for your OS platform. The INSTALL file in the src directory includes instructions for compiling and installing BIND for your OS.

BIND 8.x uses a different configuration filename than BIND 4.x uses. In BIND 4.x, the configuration file is named.boot, and in BIND 8.x, the configuration file is named.conf. This file specifies the location of zone files, sets global and zone-specific parameters, and instructs the BIND server to read and load zone files when the named daemon starts. This file is often in the /etc directory. BIND 8.x also uses a different syntax in named.conf than BIND 4.x uses in named.boot. Listing 1 shows an example of a BIND 8.2.2 named.conf file. If you upgraded from BIND 4.x to BIND 8.2.2, you need to convert named.boot to named.conf. Fortunately, BIND 8.2.2 offers a bind-bootconf shell script in the src/bin/namedbootconf directory to convert named.boot to named.conf without a manual modification.

By default, BIND 8.2.2 disables dynamic updates. You must enable the dynamic update function per zone. To enable dynamic updates in a zone, you need to add an allow-update {address\_match\_list} statement to the zone's section of BIND's named.conf file. You can use an existing zone or create a new zone based on your AD design. The IP addresses in the allow-update address match list tell the system which hosts in the list to authenticate for dynamic updates. For example, in Listing 1, named.conf allows only hosts in the subnet 192.168.1/24 to dynamically update the forward lookup zone acme.com and reverse lookup zone 1.168.192.in-addr.arpa. However, this authentication method doesn't fend off attacks. Attackers can use a host listed in the address match list to attack a dynamic zone by sending a dynamic update request that deletes an important record or all records in the zone. To ensure DNS data authentication and integrity, RFC 2065 implemented DNS Security (DNSSEC) extensions in BIND 8.2.2. Unfortunately, Win2K doesn't support DNSSEC, so you can't use DNSSEC in BIND 8.2.2 in a Win2K deployment.

**Listing 1:***A Sample BIND 8.2.2 Configuration File*

```

# zone files are stored at directory /usr/local/named
# allow zone transfer to server 198.168.1.10
# allow any computer to query this name server
options {
    directory "/usr/local/named";
    allow-transfer { 198.168.1.10; };
    allow-query { any; };
};

# this name server is the primary master of the forward zone acme.com
# the zone file is db.acme
# allow dynamic updates of the zone from the subnet 192.168.1.0
zone "acme.com" {
    type master;
    file "db.acme";
    allow-update { 192.168.1/24; };
};

# this name server is the primary master of the reverse zone 1.68.192.in-addr.arpa
# the zone file is db.192.168.1
# allow dynamic updates of the zone from the subnet 192.168.1.0
zone "1.68.192.in-addr.arpa" {
    type master;
    file "db.192.168.1";
    allow-update { 192.168.1/24; };
};

# specify the root hint file that contains Internet root DNS server addresses zone "." {
    type hint;
    file "root.cache";
};

# forward zone for localhost
zone "localhost" {
    type master;
    file "localhost";
};

# reverse zone for localhost
zone "0.0.127.in-addr.arpa" {
    type master;
    file "db.127.0.0";
};

```

The format of a zone file in BIND 4.x and BIND 8.x is the same. However, to support negative caching of DNS queries (functionality that RFC 2308 standardized), BIND 8.2 and later use the seventh parameter in a System Object Access (SOA) record (i.e., minimum TTL) as an explicit negative caching TTL parameter. If a DNS resolver tries to query a nonexistent record, the negative caching TTL parameter tells the DNS resolver how long to cache the information stating that the Resource Record (RR) doesn't exist. BIND 8.1 and earlier versions that don't support RFC 2308 traditionally use the minimum TTL parameter as the default positive caching TTL value for RRs without an explicit TTL value. This positive caching TTL parameter lets a DNS resolver cache a resolved record in the amount of time that the parameter specifies. When you use an old zone file (i.e., a BIND 8.1 or earlier zone file) for BIND 8.2.2, you need to add a \$TTL directive (e.g., \$TTL 86400) to specify a default positive caching TTL value in the zone file. The \$TTL directive placed before the SOA record becomes the default positive caching TTL value for all records in that zone. If you don't add

a \$TTL directive, BIND 8.2.2 uses the negative caching TTL value as the positive caching TTL value, and the negative caching TTL value is usually much smaller than the positive caching TTL.

After you make necessary changes to the `named.conf` file and zone files, you can start or restart the `named` daemon so that it uses the new configuration and zone files. You might want to use the BIND `Nsupdate` utility first to ensure that your DNS server supports dynamic updates. After you verify your server's support, your BIND DNS server is ready to accept dynamic updates from Win2K systems. When BIND receives a dynamic update, it saves the update in a log file whose name corresponds with the zone name (e.g., `db.acme.log`) and updates the DNS data in memory. BIND updates the zone file periodically (about once an hour) and removes the old log file when BIND commits the update. In this way, BIND aggregates dynamic updates to a batch process rather than updating the zone file every time BIND receives a dynamic update. This mechanism can simultaneously handle numerous updates.

When you use dynamic updates for a zone in BIND, don't manually modify the zone file. BIND sets the zone file attribute to read-only after it updates the zone, so any manual changes you make between zone updates will be lost. If you need to add, remove, or modify a record, you can use the `Nsupdate` utility to manually send a dynamic update request to the zone.

BIND 8.2 and later support incremental zone transfer (RFC 1995 defines this functionality), which lets a secondary name server (a slave name server in BIND 8.x terminology) transfer only changes rather than an entire update. The BIND master server maintains the changes between one SOA serial number and the next in a zone's `.ixfr` file (e.g., `db.acme.ixfr`). When a slave server requests an incremental zone transfer, the master server transfers only the changes. Win2K DNS supports incremental zone transfer.

## Ready, Set, Go

Now that you have a better understanding of Win2K's DNS requirements, you can decide which option will work best to integrate your UNIX DNS service and Win2K. After you set up a functional DNS infrastructure that supports SRV record and dynamic updates, you're ready to deploy Win2K and AD. This setup lets you take advantage of Win2K's enhancements and build your network applications around AD.

## Chapter 9

# Maximizing BIND DNS Security

—by *Tao Zhou*

No company can afford to ignore the security of its DNS service, the crucial service that provides host-name and IP address resolution on the Internet. Although Windows 2000 provides a built-in DNS service, Berkeley Internet Name Domain (BIND) is the most widely used DNS software on the Internet, and many Win2K administrators use BIND to maintain their Internet DNS servers. You can use BIND access-control settings to implement a basic layer of protection. However, those settings don't address two important security concerns: data authentication and data integrity. To be truly secure, a DNS client or server must communicate only with a trusted DNS server and must authenticate received data. Authentication involves confirming that no one has intercepted a query reply and changed its content during its transmission over the Internet.

To help companies guarantee data authentication and integrity, the Internet Engineering Task Force (IETF) launched development of the DNS Security (DNSSEC) Internet standard, which uses public key and digital signature technologies that developers can apply in their DNS software implementations. The IETF also developed the Transaction Signature (TSIG) DNS transaction-authentication protocol, which uses shared secret key technology to help secure DNS transactions. (IETF Request for Comments—RFC—2535 and RFC 3008 discuss DNSSEC; RFC 2845 deals with TSIG. For more information about DNSSEC, you can also visit NLnet Labs' DNSSEC resources Web site at <http://www.nlnetlabs.nl/dnssec>.)

The Internet Software Consortium (ISC), which develops and maintains BIND source codes, has incorporated DNSSEC and TSIG in BIND 8.2.3 and later and in BIND 9.1.0 and later. The newer BIND 9 versions have better support for DNSSEC and TSIG than do the BIND 8 versions. Both generations can generate public and private keys, but BIND 9.1.0 has extended zone-signing capabilities. Therefore, this chapter focuses on implementing DNSSEC and TSIG in BIND 9.1.3. (The ISC hasn't imported BIND 9.2.0 or later into Win2K or Windows NT yet, but you can run BIND 9.1.0 on most UNIX and Linux platforms. Also, BIND 9.2.2 supports Win2K and NT.) This chapter assumes you have a basic knowledge of BIND, digital signature technology, and shared secret key technology.

## DNSSEC Basics

Implementing DNSSEC on your BIND DNS server involves signing the server's DNS domain zone file. (I explain that process later.) DNSSEC then uses digital signature technology to sign each record in the zone. When a DNS client queries a specific DNS record, the DNS server replies with the signed version of the requested record. The DNS client then uses digital signature technology to confirm the data integrity of the received record and to authenticate the DNS server. To support digital signature technology, DNSSEC introduces two new DNS Resource Records (RRs): SIG and KEY. The security protocol also introduces the NXT RR to help authenticate nonexistent DNS records.

## The SIG RR

A SIG RR contains the digital signature for each original RR (e.g., Start of Authority—SOA, NS, A, PTR, MX, CNAME). In a signed zone, DNSSEC associates each original RR with a SIG RR to form a record set. Listing 1 shows the zone file for an unsigned, unsecured zone, `us.example.com`. Figure 1 shows the zone file for the corresponding secure, signed zone. As Figure 1 shows, a SIG RR follows each original RR in the signed zone. A SIG RR's syntax is

```
corresponding_RR_type
  signing_algorithm
  number_of_owner_labels
  original_TTL signature_expiration
  signature_inception key_tag signer signature
```

For example, callout D in Figure 1 shows the SIG RR for `www.us.example.com`'s A record. This output shows that the SIG RR corresponds to an A RR and that DNSSEC used Digital Signature Algorithm (DSA) to sign the original RR. (RFC 2535 discusses the possible signing algorithms: 1 is RSA/Message Digest 5—MD5, 2 is Diffie-Hellman, and 3 is DSA.) The A RR's owner (i.e., `www.us.example.com`) has four labels (i.e., `www`, `us`, `example`, and `com`). The original Time to Live (TTL) value of the A RR is 86400 seconds (i.e., 1 day), the signature expires at 6:10:13 Greenwich Mean Time (GMT) on April 3, 2001 (i.e., 20010403061013), and the signature became valid at 6:10:13 GMT on March 4, 2001 (i.e., 20010304061013). The key tag (i.e., a number that identifies the zone's public and private key pair) is 51297. The signer is `us.example.com`, and the final strange string is the signature.

### LISTING 1:

#### *Sample Unsecured Zone*

```
$TTL 1D
$ORIGIN us.example.com.
@ IN      SOA      us.example.com.  root.us.example.com. (
    2001030301
    2H
    1H
    1W
    1D )
IN       NS       ns1.us.example.com.
ns1     IN       A       192.168.1.10
mail    IN       MX      10      www.us.example.com.
www     IN       A       192.168.1.11
(A) $INCLUDE  Kus.example.com.+003+51297.key
```

**Figure 1:**  
*Sample signed zone*

```

; dnssec_signzone version 9.1.0
us.example.com. 86400 IN SOA us.example.com. root.us.example.com. (
2001030301 ; serial
7200 ; refresh (2 hours)
3600 ; retry (1 hour)
604800 ; expire (1 week)
86400 ; minimum (1 day)
)
86400 SIG SOA 3 3 86400 20010403061013 (
20010304061013 51297 us.example.com.
CCpoka7cWxbbZA2bWXDKX3oXdfgLb30oout1
N0GcLr6Ddt0Cj4y+XZk= )
86400 NS ns1.us.example.com.
86400 SIG NS 3 3 86400 20010403061013 (
20010304061013 51297 us.example.com.
CDMXm3yoT2w54Lh6c1005X2B57LLAC5gjTH6
N0oal5EsQRx04mr1V2s= )
(A) 86400 KEY 256 3 3 (
CJHScmcPQm31yH4H0iDgBZWHT1LB7Vy02kOT
uJCDmEDLfTmKJTJiB0DxCVnDMhipviVlv+g
Bx1h4/OtMAC8BsXTorwQqZiY7FDgpP1f6+Zg
E4OPbn3TNHw11TFGAmjKf0Y98sJ/uq1uFRAH
VFeDZysCAevcMuG7VjPeQ0eHamQFe1Wd0JmK
vKt4iudLNdzdNesRcJEmORCKpLgDne9nLyGE
B7tWAarq7kTHFCjyppMBt1yC1FtmD5hJ2U31
WPZyErWQnDMCSV65ykDNDXXhk9sAH2WSjJCE
pGbLfZXBJuVQFAPc96hSaBdDhRsLW5+ZjV77
0eLaVa+o85TVjBpBYW/xLRZ9ffe0DKdYLT1P
SdB2zKcSe094qjuXd9y4D7qVxs3RaZfcm4Zw
s38LhyLfvUUJMaI24EbPXFLaqiS1ggStf7rL
80UVH4Zv1ZcsDQRcM7LmhGnGENJwq1Zk12rS
Y2KoSpfasxwZyzGxf5AQ1vLOHRw0Z41DaTnp
R+/Aaogp0jYcYyMLLG/iP+V1zhImVpMKrJuG ) ; key id = 52324
(B) 7200 SIG KEY 3 3 7200 20011231235959 (
20010304053427 47063 example.com.
CF8nfzUST+NAxh3PWAkZWylkWp93D32pYIhw
jjAQzVxr0e839TpWo7c= )
86400 NXT mail.us.example.com. NS SOA SIG KEY NXT
86400 SIG NXT 3 3 86400 20010403061013 (
20010304061013 51297 us.e 86400 SIG NXT 3 4
86400 20010403061013 (
20010304061013 51297 us.example.com.
CDgquyVkwz175PwtMZQ4NN90vH2KfW4tsEt
hJwvgkakqAL8s/PqMLw= )
ns1.us.example.com.
86400 IN A 192.168.1.10
86400 SIG A 3 4 86400 20010403061013 (
20010304061013 51297 us.example.com.
CB+byYi5uAaVYSql8NA1aEKqA070UgApAWP2
Q5SF4DJtEjv/5HtXPt0= )
(C) 86400 NXT www.us.example.com. A SIG NXT
86400 SIG NXT 3 4 86400 20010403061013 (
20010304061013 51297 us.example.com.
CIJ3JKWo2NXNJGET8etRH3zCwgJCWy06vQ3N
zydr1CXsSvNjikh0K0E= )
www.us.example.com.
86400 IN A 192.168.1.11
(D) 86400 SIG A 3 4 86400 20010403061013 (
20010304061013 51297 us.example.com.
CIxQaCMNQHXq9HP+r5hW3S/I/YH0d0DMKq+
NN8APyzW12Svq23Z078= )
86400 NXT us.example.com. A SIG NXT
86400 SIG NXT 3 4 86400 20010403061013 (
20010304061013 51297 us.example.com.
CD4KEvDhYmzAggi8eQVw956eK2hddroRsQ0M
Yaxcof8o13zVkybYHSM= )

```

## The KEY RR

You use a zone's private key (from its public and private key pair) to sign the zone during DNSSEC implementation. You keep the private key secret, and the KEY RR stores the public key, which a DNS client uses to verify the signatures of received DNS records. DNSSEC provides a corresponding SIG RR for each KEY RR, but the parent zone signs the KEY RR. For example, the parent zone example.com would sign a KEY RR in the signed zone us.example.com. The KEY RR's syntax is

```
flags protocol algorithm public_key
```

Flags indicate the type of the included key and how to use it. For example, callout A in Figure 1 shows us.example.com's KEY RR. This output shows that the key is a zone key and is used only for authentication. The protocol is DNSSEC (represented by the number 3), the signing algorithm is DSA (represented by the number 3), and the final string is the zone's public key. (RFC 2535 discusses the meanings of the KEY RR's options.) The KEY RR's corresponding SIG RR, which callout B in Figure 1 shows, defines the signer of the KEY RR (i.e., us.example.com's parent zone, example.com).

## The NXT RR

The SIG and KEY RRs are strong enough to authenticate existing DNS records, but DNSSEC wouldn't be truly secure without a mechanism to authenticate nonexistent records. For example, consider the unsecured www.us.example.com zone file that Listing 1 shows. If a client queries a nonexistent record, product.us.example.com, the DNS server running the unsecured zone will return the SOA RR and an NXDOMAIN error, which indicates that the queried record doesn't exist.

If DNSSEC used the same method to reply to such a query—even if DNSSEC provided a corresponding SIG RR for the signed SOA RR—an intruder could easily wreak havoc. The intruder could query the zone or listen to the wire to determine us.example.com's SOA and corresponding SIG RRs. Then, when a client queried the DNS server for the existing record www.us.example.com, the intruder could send that client the SOA and SIG RRs and the NXDOMAIN error before the DNS server could respond. The client would verify the SOA RR's signature and determine that www.us.example.com didn't exist.

To prevent such problems, DNSSEC introduced the NXT RR, which indicates the record-owner names that don't exist within a specified name interval. (For example, if a client knows that within the name interval 1 through 4, the current record is 1 and the next record is 4, the client can infer that the records 2 and 3 are nonexistent.)

DNSSEC arranges a signed zone's record sets in canonical order (i.e., zone name, wildcard—\*—record—if any, then the other records in alphabetical order) and inserts a NXT RR for each record. A NXT RR's syntax is

```
next_record_owner record_types_
in_current_set
```

For example, ns1.us.example.com's NXT RR, which callout C in Figure 1 shows, states that the next record's owner name is www.us.example.com and that the current record (i.e., ns1.us.example.com) contains the RR types A, SIG, and NXT. From this NXT RR, the client can determine that no other records exist between ns1.us.example.com and www.us.example.com. A zone's final NXT RR always

points back to the zone's first record (i.e., the zone name). DNSSEC signs each NXT RR with an associated SIG RR, and the NXT record's syntax prevents intruders from using the NXT RR to make a client believe an existing record doesn't exist.

## Using DNSSEC in BIND

To use DNSSEC in BIND 9.1.3, you must set up a secure zone on a BIND 9.1.3 server. This process involves four steps: generating keys, creating a keyset, signing the child zone's keyset, and signing the zone.

### Generating Keys

A secure zone must have one or more public and private key pairs, called zone keys. You use the zone's private key to sign the zone. DNS clients use the corresponding public key to verify signatures. When the parent zone (e.g., example.com) delegates a secure child zone (e.g., us.example.com), you can use the parent zone's private key or keys to sign the child zone's key or keys. If the parent zone has multiple key pairs, you can use different keys to sign the parent and child zones' key or keys.

You use the `dnssec-keygen` command to generate keys on a BIND 9.1.3 server. (Enter the command without any parameters to see command-usage information.) For example, to generate a key pair for the child zone `us.example.com` in Listing 1, enter the following command on the BIND server:

```
dnssec-keygen -a DSA -b 1024 -n ZONE us.example.com.
```

The `-a` option defines which digital signature algorithm the generated keys will use. According to RFC 2535, DSA is the mandatory algorithm for DNSSEC interoperability, but BIND 9.1.3 also supports Diffie-Hellman, Hash Message Authentication Code (HMAC)-MD5, RSA, and RSA/MD5. The `-b` option specifies the key length (1024 bits in this example). The `-n` option defines the keys' name type, which can be zone, host, entity, or user. The final parameter in the sample command is the key name, which must be the same as the zone name if the specified name type is zone.

The `dnssec-keygen` command also lets you choose the protocol that the generated keys will support; DNSSEC is the default. The command automatically saves the generated keys in two output files in the current directory; the parameters you use in the command determine the filenames. The sample command would store the public key in the file `Kus.example.com.+003+51297.key` and store the private key in the file `Kus.example.com.+003+51297.private`. The first number set represents the digital algorithm (in this example, DSA, which DNSSEC represents by the number 3). The second number set (51297 in this example) is the key tag, which distinguishes the keys from other key pairs and appears in the signed zone's SIG records. You need to protect the private key file and keep it secret, but you need to include the public key filename in the child zone's unsecured zone file, as callout A in Listing 1 shows.

### Creating a Keyset

Before the parent zone can sign a child zone's public key, you need to create a keyset containing the child zone's public key that the parent zone can recognize and sign. You use the `dnssec-makekeyset` command to create this keyset. For example, the command

```
dnssec-makekeyset -t 7200 -e 20011231235959 Kus.example.com.+003+51297
```

creates a keyset for the child zone `us.example.com`. The `-t` option defines the SIG and KEY records' TTL value. The sample command specifies the TTL as 7200 seconds; 3600 seconds is the default. The `-e` option defines the end time of the parent zone's signature. The sample command specifies that the signature expires at 23:59:59 on December 31, 2001 (i.e., 20011231235959); the default signature end time is 30 days after the start time. (The sample command accepts the default signature start time, which is now, but you can use the `-s` option to specify a start time.) The command saves the keyset output file as `keyset-zone.` in the current directory. (The ending period is part of the filename.) For example, the output file `keyset-us.example.com.` would result from the sample command.

### ***Signing the Child Zone's Keyset***

Next, you need to submit the child zone's keyset to the parent zone so that the parent zone can sign the keyset. The parent zone uses the `dnssec-signkeyset` command to sign a child zone's keyset. For example, to instruct the parent zone `example.com` to sign the child zone `us.example.com`'s keyset, use the command

```
dnssec-signkeyset keyset-us.example.com. Kexample.com.+003+47063
```

By default, the command uses the signature start and end times you specified when you created the child zone's keyset, but you can use the `-s` and `-e` options to overwrite those times. The final parameter is the parent zone's key tag. The command saves the signed keyset output file as `signedkey-zone.` in the current directory. (The ending period is part of the filename.) For example, the output file `signedkey-us.example.com.` would result from the sample command. The parent zone then needs to securely ship this output file to the child zone; store the output file in the same directory as you store the child zone's keys.

### ***Signing the Zone***

Now, you can sign the child zone. To do so, you use the `dnssec-signzone` command. For example, to sign the unsigned child zone `us.example.com` that Listing 1 shows, use the command

```
dnssec-signzone -o us.example.com /usr/local/etc/db.us.example
```

The `-o` option specifies the zone name (i.e., `us.example.com`). The second parameter is the full path of the zone file (i.e., `/usr/local/etc/db.us.example`). This sample command creates an output file named `/usr/local/etc/db.us.example.signed`. To instruct the BIND 9.1.3 DNS server to load the secure zone file, you need to include the signed zone filename in `us.example.com`'s `named.conf` file. If you update a zone file after you've signed the zone, you need to resign the zone.

DNSSEC provides reliable security for DNS transfers. The sidebar "A Secure Transaction" presents examples of such transactions for the DNSSEC-secured zone that Figure 1 shows.

## **Using TSIG in BIND**

BIND's DNSSEC implementation is powerful, but to be effective, it requires DNSSEC-enabled clients and a public key infrastructure (PKI) on the Internet. But Windows client computers don't support DNSSEC, and the Internet doesn't have a PKI infrastructure in place. Until the Internet can more fully support DNSSEC, you can use TSIG to authenticate DNS transactions between two hosts.

## A Secure Transaction

When a DNS client queries the sample record `www.us.example.com`, the DNS Security (DNSSEC) server running the secure zone `us.example.com` sends the client `www.us.example.com`'s original A Resource Record (RR) with its corresponding SIG RR and the zone's KEY RR with its SIG RR. If the client is DNSSEC-enabled, it first checks the signature-validation time in the A RR's corresponding SIG RR. If the signature starting time is later than the current time or if the signature expiration time is earlier than the current time, the client rejects the A RR. If the signature-validation time is acceptable, the client then uses the public key and the algorithm in the KEY RR to verify the signature in the A RR's SIG RR.

After validating the A RR's signature, the client verifies `us.example.com`'s public key signature (which the client used to check the integrity of the A RR). The parent zone `example.com` signed `us.example.com`'s public key, so the client requests `example.com`'s public key (if the client doesn't already have the key locally or in its cache). If the client trusts `example.com`'s public key and thus verifies the validity of `us.example.com`'s public key, the client then accepts that the query response is truly from `us.example.com`.

The verification of the signer's public key can be recursive until the client finds a trusted signer. In DNSSEC, this trusted signer could be the Internet root zone running on the Internet root DNS servers. Therefore, you should preconfigure at least one trusted key in a DNSSEC-enabled client's local computer. This key can be the Internet root zone's public key if the client trusts only the root zone.

Suppose a client queries the signed zone `us.example.com` for a nonexistent record, `product.us.example.com`. The zone `us.example.com` doesn't contain that record, so the DNS server instead sends the client the record `ns1.us.acme.com`, which is the record that falls alphabetically before the requested nonexistent record. The NXT record in `ns1.us.example.com` indicates that the next record is `www.us.example.com`. Thus, the client can ascertain that no record exists between `ns1.us.example.com` and `www.us.example.com`—`product.us.example.com` doesn't exist.

In TSIG, two hosts (i.e., two DNS servers or a DNS server and a DNS client) share a secret key and use an MD algorithm to authenticate a DNS message transaction. TSIG doesn't use presigned zone files, as DNSSEC uses. When a TSIG-enabled host sends a message packet, the host generates a signature for the packet. The signature is good only for that transaction. TSIG's shared secret key technology means that multiple hosts share the same secret key. This type of key is easier to compromise than the public and private keys that DNSSEC uses, so TSIG isn't as secure as DNSSEC.

You can use TSIG to authenticate such DNS message transactions as query requests and responses, dynamic updates, and zone transfers. For example, you can use TSIG between your DNS servers and your ISP's or business partners' DNS servers to secure those communications. If your BIND server runs a dynamic DNS (DDNS) zone, you can use TSIG to authenticate dynamic updates from clients, DHCP servers, and other servers that support TSIG. This type of authentica-

tion for dynamic updates is more secure than IP address authentication. You can also use TSIG to authenticate zone transfers from the primary DNS server to secondary DNS servers. BIND 8.2.4 and 9.1.3 primarily support TSIG for server-to-server communication. These BIND versions' dynamic update tool, Nsupdate, also supports TSIG through use of a secret key option.

Suppose you want to use TSIG between two hosts, host1 and host2. You first need to generate a secret key for the hosts. To do so, you can use the `dnssec-keygen` command

```
dnssec-keygen -a HMAC-MD5 -b 128 -n HOST host1-host2.
```

As I explained earlier, this command's `-a` option defines the digital signature algorithm that the generated key will use. RFC 2845 specifies HMAC-MD5 as the mandatory algorithm for TSIG interoperability, and BIND supports only HMAC-MD5 for TSIG. You can use a key length as long as 512 bits for HMAC-MD5. (The example uses a key length of 128 bits.) The key name in this example is `host1-host2.` (the ending period is part of this name). The output private key file contains the generated secret key. For example, the sample command creates the private key file `Khost1-host2.+157+59290.private`. The `Key:` line in that file contains the secret key. You need to securely deliver this key to both hosts.

If `host1` and `host2` are both BIND servers, you need to add the key statement that Listing 2 shows to both servers' `named.conf` files. To protect the key statement, you must restrict read access to `named.conf` so that only the root account can read the file.

A better option is to maintain a separate file that contains only key statements, protect that file, then include that filename in `named.conf`. That way, you don't need to restrict `named.conf`.

To permit `host1` (IP address 192.168.1.10) to use the key and TSIG when sending a message to `host2` (IP address 192.168.2.10), add the server statement that Listing 3 shows to `host1`'s `named.conf` file. Add the server statement that Listing 4 shows to `host2`'s `named.conf` file.

If `host2` dynamically updates a dynamic zone on `host1`, you can use the TSIG key to authenticate `host2` and its request. To do so, use an `allow-update` statement, which Listing 5 shows, in `host1`'s `named.conf` file.

### LISTING 2:

#### *Sample Key Statement in Named.conf*

```
key host1-host2. {
    algorithm hmac-md5;
    secret "Rfya3QvNbZ/o9YL8/qNgKdv++3iIqCoShp2rRA==";
};
```

### LISTING 3:

#### *Sample Server Statement in Host1's Named.conf*

```
server 192.168.2.10 {
    keys { host1-host2. ;};
};
```

**LISTING 4:***Sample Server Statement in Host2's Named.conf*

```
server 198.168.1.10 {
    keys { host1-host2. ;};
};
```

**LISTING 5:***Sample Allow-Update Statement in Named.conf*

```
allow-update { key host1-host2. ;};
```

## Looking Forward

DNS is a fundamental Internet service. Any Web services that require host-name resolution depend on DNS. Without a secure DNS service, these Web services aren't secure. Although DNSSEC isn't yet widely deployed on Internet clients, BIND's rich set of security tools, including DNSSEC and TSIG, can help you secure your DNS service. Fault-Tolerant Mesh of Trust Applied to DNSSEC (FMESHHD), a Defense Advanced Research Projects Agency (DARPA) project with the objective of deploying DNSSEC on the Internet, is building a DNSSEC testbed for organizations to use. (For more information about this testbed, visit <http://fmeshd.nge.isi.edu>.) You can also use DNSSEC within your intranet without needing to establish a trust relationship with other companies.

TSIG can help you secure transactions with hosts that don't yet support DNSSEC, and several other IETF protocols can enhance TSIG's capabilities. The Secret Key Establishment for DNS (TKEY) RR, which IETF RFC 2930 describes, automates TSIG secret key generation and exchange. The DNS Request and Transaction Signatures, or SIG(0), which IETF RFC 2931 describes, uses public keys to authenticate DNS requests and transactions. BIND 9.1.0 through 9.1.3 have some limited TKEY and SIG(0) functions but provide no documentation for these functions. I expect TKEY and SIG(0) to be fully implemented in a future BIND version.

At the time of this writing, Win2K's DNS service doesn't support DNSSEC but offers limited support for TSIG. (Windows Server 2003's DNS has basic support for DNSSEC but as a secondary DNS server. For more information, see "Using DNS Security Extensions (DNSSEC)" at [http://www.microsoft.com/resources/documentation/IIS/6/all/proddocs/en-us/sag\\_DNS\\_imp\\_DNSSECsupport.asp](http://www.microsoft.com/resources/documentation/IIS/6/all/proddocs/en-us/sag_DNS_imp_DNSSECsupport.asp).) Win2K DNS supports secure dynamic updates in an Active Directory (AD)-integrated dynamic DDNS zone. Microsoft bases Win2K DNS's implementation of secure dynamic updates on the company's proposed IETF draft "GSS Algorithm for TSIG (GSS-TSIG)"; at the time of this writing, you can view the most recent draft at <http://www.ietf.org/mail-archive/ietf-announce-old/Current/msg17757.html>. To extend TSIG, GSS for TSIG applies the Generic Security Service API (GSS-API—see IETF RFC 2078 for information about this API).

## Chapter 10

# DNS FAQs

### *Q. How do I rename a Windows 2000 domain's DNS name?*

- A.** To rename a Win2K domain's DNS name, your domain must meet the following conditions:
- Your domain must be in mixed mode.
  - You must have at least one Windows NT 4.0 BDC or have the ability to create an NT 4.0 BDC.

Renaming the DNS name requires you to demote all Win2K domain controllers (DCs) to servers and temporarily promote the BDC to the PDC. Keep in mind the following considerations before attempting this process:

- Only the domain account information and password will be saved; extra Active Directory (AD) information will be lost.
- When you demote your Win2K DCs, you'll lose your load-balancing and fault-tolerance configurations.
- All child domains must be in mixed mode so that you can demote them to NT 4.0 domains.
- You can't change a domain's NetBIOS name.

You must demote the child domains first, starting with the lowest domains in the hierarchy (e.g., you need to apply changes to grand.child.domain.com before applying changes to child.domain.com). To demote the domains to NT 4.0, perform the following steps on each domain:

1. Perform a full backup of the domain.
2. Verify that the NT 4.0 BDC is up-to-date by typing

```
net accounts /sync
```

3. Review the system log to ensure the BDC update was successful.
4. Use Dcpromo to demote each Win2K DC in the domain until only one Win2K DC is left running.
5. Detach the last Win2K DC from the network (after you disconnect this system, you can run Dcpromo; however, you might need to plug the machine into another hub or switch for Dcpromo to function).
6. Use the Server Manager application to promote the NT 4.0 BDC to PDC. Although the application will say it can't find the PDC, click Yes to continue.

After you demote domains (child and parent) to NT 4.0, you can begin to upgrade the systems to Win2K, starting with the highest machine in the hierarchy and working down (e.g., you need to apply changes to domain.com first, apply changes to child.domain.com second, and finally apply changes to grand.child.domain.com). To rename the DNS server, upgrade the NT 4.0 PDC first,

giving it the new DNS name. Then, run Dcpromo again on all other Win2K servers that were previously DCs and associate them with the new DNS domain.

This process is highly complex and time consuming, and Microsoft doesn't recommend that you attempt this procedure. If you're considering renaming your DNS server, keep in mind that Windows .NET Server (Win.NET Server) lets you use the random.exe utility to rename the DNS server. So, you might want to consider upgrading to Win.NET Server, which also lets you rename the NetBIOS domain name.

—by John Savill

***Q. How do I stop my Windows domain controllers (DCs) from dynamically registering DNS names?***

**A.** By default, the Netlogon service on a DC registers dynamic DNS (DDNS) records to advertise Active Directory (AD) Directory Service (DS) services. However, you can edit the registry to disable this feature.

1. Start regedit.exe.
2. Go to the HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\Netlogon\Parameters registry subkey.
3. If the UseDynamicDns value exists, double-click it, and set it to 0.
4. If the UseDynamicDns value doesn't exist, from the Edit menu, select New, DWORD Value.
5. Enter a name of UseDynamicDns and press Enter.
6. Double-click the new value, and set it to 0.
7. Click OK.
8. Close regedit.
9. Reboot the server.

If you disable DDNS updates, you should manually create the necessary records based on the %windir%\system32\config\netlogon.dns file.

—by John Savill

***Q. How do I stop DNS cache pollution?***

**A.** DNS cache pollution can occur after DNS spoofing. Spoofing is the sending of nonsecure data in response to a DNS query. Spoofing can be used to redirect queries to a rogue DNS server and can be malicious in nature. You can configure Windows NT DNS to filter out responses to unsecured records.

1. Start regedit.exe.
2. Go to the HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\DNS\Parameters registry subkey.
3. From the Edit menu, select New, DWORD Value.
4. Enter a name of SecureResponses and press Enter.
5. Double-click the new value and set to 1. Click OK.

For more information about the SecureResponses value, see the Microsoft article “Microsoft DNS Server Registry Parameters, Part 2 of 3” at <http://support.microsoft.com/?kbid=198409>.

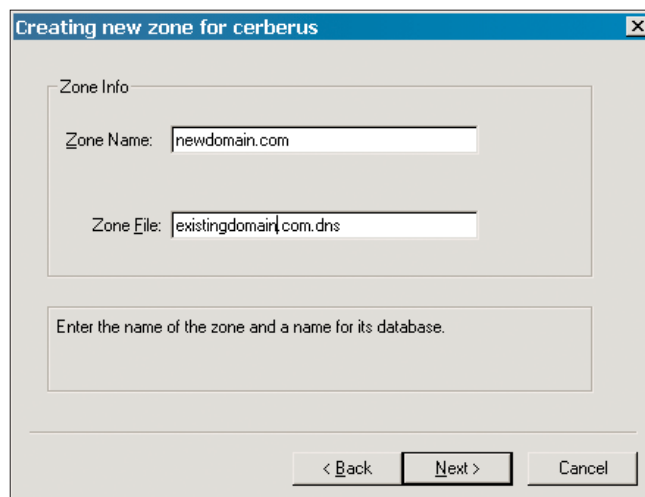
—by John Savill

**Q. How do I clone Windows DNS zone files?**

**My company has several subsidiary companies, all of which have unique company and DNS domain names. These companies all use the same server for their Internet-accessible services (e.g., Web servers, FTP servers, DNS servers). Therefore, I often need to create new DNS zone files that are essentially identical to those that already exist, with the exception of the domain name portion (e.g., mycompany.com). Using Windows NT's DNS Manager utility to recreate these files from scratch is tiresome. Do you know of any tricks I can use to speed up this process?**

**A.** To easily duplicate DNS zone files and substitute the correct domain name for the new zone file, you can trick NT's DNS Manager. To duplicate an existing zone file and its record contents to a new zone, run DNS Manager and begin creating the new domain and zone file (i.e., select the server name and choose New Zone from the DNS menu). This process launches the Create New Zone wizard. In the first dialog box, the wizard asks you to choose whether this zone is primary or secondary. Click Primary, then Next. The second dialog box, which Figure 1 shows, asks you to name the domain and provide the name of the DNS zone file that contains the records. To trick DNS Manager, type the new domain in the first text box but override the default zone name in the second text box (e.g., newdomain.com.dns) with the name of the existing zone file for the domain you want to duplicate (e.g., existingdomain.com.dns).

**Figure 1:**  
*Tricking DNS Manager*



After you select Finish to complete the wizard, DNS Manager will have created a new zone file for the new domain name. However, DNS Manager also will have automatically copied all records from the existing zone file and renamed all records that reference the root domain name (e.g., SOA, A, MX) so that they now reference the new domain name. Although you still need to check the data values in the right column to ensure that they're accurate for each record in the new domain, this handy tip lets you easily copy zone data from one domain to another through the DNS Manager GUI.

—by Sean Daily

***Q. What is the proper use of CNAME records?***

***My company locally hosts its own DNS, but our ISP recently discovered problems in the configuration of the DNS zone file that supports our domain name. The ISP said we shouldn't use Canonical Names (CNAMEs) for our two DNS servers. In our current setup, we use CNAMEs for two DNS servers—ns1 and ns2. I don't think we've had any name-resolution problems, so the ISP's statement confuses me. Can you shed light on this situation?***

***A.*** A Name Server (NS) record in a DNS zone file typically shouldn't point to a CNAME—just as a mail exchanger (MX) record shouldn't point to a CNAME. Although using a CNAME entry for an NS record is possible, it's not the best choice. The primary reason you should avoid using CNAME records in the MX, NS, and Source of Authority (SOA) records is performance: Internet hosts make frequent queries to these records, so you want them to require as little time and as few resources as possible to complete. If you reference a CNAME, you generate additional queries that can impede performance. Instead, consider another strategy.

First, for each DNS server, you can create two address (A) records that point to the same IP address on the server. One A record is the name you used in the SOA and NS records (e.g., ns1.ntsol.com), and the other is the real or descriptive name of the server (e.g., calvin.win2kinfo.com). These different A records associate the IP address with each name you want the host to use. This process is only necessary if you want to give the DNS server more than one name—for example, one name for the name server entries in the NS or SOA records in the zone file (e.g., ns1) and another name (e.g., Calvin) for your LAN clients.

Second, you can simply use one name for each server in the zone file, including references in the A, NS, and SOA records. This solution requires no additional lookups to resolve CNAME records to their backing A records. You could still use CNAME records to create additional aliases for these hosts, as long as you remember to reference only the A record name in the MX, NS, and SOA records. Figure 2 shows an example of this method.

—by Sean Daily

**Figure 2:**  
Using only one name for each server in the zone file

```

;
; Database file win2kinfo.com.dns for win2kinfo.com zone.
; Zone version: 1996061147
;
@           IN  SOA  cerberus.win2kinfo.com. administrator.w
                1996061147 ; serial number
                21600    ; refresh
                900     ; retry
                3456000  ; expire
                3600    ) ; minimum TTL
;
; Zone NS records
;
@           NS   calvin
@           NS   hobbes
;
; Zone records
;
@           MX   10   mail.myisp.net.
@           MX   0    mail

```

***Q. Why does an Active Directory (AD)-integrated DNS server take longer to start than a typical zone-based DNS server?***

**A.** Windows 2000 and later OSs can store DNS information in AD if the DNS server is a domain controller (DC). Alternatively, the OS can store DNS information on a standard primary zone-based DNS server, which is file based.

When the DNS service starts, it loads all zone information into a memory cache, regardless of whether the OS maintains the DNS information in AD or in a file. DNS information stored in a standard primary zone (i.e., read from a zone file) will load faster than information stored in an AD-integrated zone because the integrated zone must read all its records from AD. This difference in performance is simply an effect of reading information from different media (file versus AD).

—by John Savill

***Q. In a multiple DNS server environment, how do I configure the DNS servers to resolve both local and remote hosts?***

**A.** Windows 2000, Windows NT, and Windows 9x let you identify multiple DNS servers. So, for example, you might have a local DNS server on your network and a remote DNS server if you connect to the Internet. In this situation, if you list your local DNS server first, you might not be able to resolve remote names, and if you list the remote DNS server first, you might not be able to resolve local names.

In a multiple DNS server environment, if a client queries the first DNS server and that server doesn't respond, the client will query the second DNS server. If the first DNS server (e.g., a local DNS server that doesn't know about a remote host) responds with an unknown host, then the

client won't query other DNS servers. Instead, the client will resort to using other methods (e.g., LMHOSTS, WINS) to resolve the domain name.

To work around this problem, you need to configure your machines to forward DNS information, which typically means configuring local DNS server information on the clients and configuring the local DNS servers to forward unknown requests to the remote DNS servers.

—by John Savill

***Q. How do I tell which boot method my DNS service uses?***

**A.** Windows can store DNS start-up information in one of three possible locations. The OS uses the following values to record the location in the DNS BootMethod registry subkey:

- 1—In a BIND-style file (\%SystemRoot%\System32\DNS\Boot); the system still checks the registry for missing data in the file. Windows 2000 DNS doesn't support this option.
- 2—In the registry.
- 3—In both the registry and Active Directory (AD).

To check the BootMethod registry subkey to see which location your DNS service uses, perform the following steps (don't modify the value):

1. Start regedit.exe.
2. Go to the HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\DNS registry subkey.
3. View the BootMethod value and compare it to the three options listed above.
4. Close regedit.

Note that if you upgrade from Windows NT 4.0 to Win2K, you need to check this value after the upgrade. If you haven't applied Service Pack 4 (SP4) to NT 4.0, the system should use an alternative key, EnableRegistryBoot, until the DNS service populates the correct key.

—by John Savill

***Q. How do I create a caching-only DNS server?***

**A.** A DNS server typically holds records about various DNS zones. These records replicate between other DNS servers (via the Active Directory—AD—in a AD-enabled zone).

Caching-only DNS servers don't host any zones and aren't authoritative for any domains. These DNS servers simply cache results from client queries. If a client asks a caching-only DNS server to resolve [www.savilltech.com](http://www.savilltech.com), the caching-only server will query a zone-holding DNS server to resolve the domain name and the caching-only server will cache the answer. If another client then asks the caching-only server to resolve the same record, the caching server can answer from its cache. This process is similar to when a proxy server caches popular Web pages.

Caching servers are useful for sites connected through a WAN with a local caching-only DNS server. Name-resolutions requests can occur locally and therefore save on network traffic.

To configure a caching-only DNS server, perform the following:

1. Ensure the machine has a static IP address.
2. Install the DNS service as normal. Start the Control Panel Add/Remove Programs applet, and select Add/Remove Windows Components, Networking Services, Details, Domain Name System (DNS), OK, Next, Finish.
3. Start the Microsoft Management Console (MMC) DNS snap-in.
4. From the Action menu, select *Connect to DNS Server*.
5. In the Select Target Computer window, select the *The following computer* option and enter the name of a DNS server you want to cache.
6. Click OK.

The machine will now start to gather a cache of host-IP mappings. To clear the cache, right-click the server name (not the server the DNS service is caching from) and select Clear Cache from the context menu.

—by John Savill