Winchester Systems

White Paper

FlashNAS™ Unified Storage

Affordable Feature-Rich NAS

Supports CIFS & NFS for Windows, Linux, Unix & MAC OS

September 2012

101 Billerica Ave, Bldg. 5, Billerica, MA 01862 781-265-0200 www.winsys.com

WINCHESTERSYSTEMS®
Purpose-Built Storage

Executive Summary

In this day and age, almost every organization uses computer applications running on a mix of operating systems (OSes): usually either a version of Microsoft Windows or some flavor of Unix/Linux. Most Unix applications share files over the network using the Network File System (NFS) protocol. Windows applications, on the other hand, almost always use Server Message Block (SMB, also known as Common Internet File System or CIFS, which will be used throughout this white paper moving forward).

At the same time, IT shops are continually asked to do more with less. A typical imperative: more efficient use of server and storage resources. As a result, consolidation has become a normal part of operating just about any data center with more than a handful of servers. Consolidating storage means accessing data over a network from multiple servers—regardless what OS they run. So it's no surprise multiprotocol support has become a battle ground for leading NAS vendors. But the way their products handle the idiosyncrasies of Unix and Windows security is also a huge factor in keeping IT complexity—and therefore costs—under control.

How does one go about building a multiprotocol NAS server? A popular technique among low-cost vendors and do-it-yourselfers is to cobble together open-source software packages atop Linux to run on an industry-standard server. But it comes with support and interoperability risks. At the other extreme, vendors such as EMC and NetApp build systems from the ground up, implementing their own OS kernel, file system, networking, NFS protocol, and Windows CIFS stack. Such products offer high-end capabilities—but at high-end prices. And as they add more and more features and functions, customers face increased operational complexity.

Winchester Systems FlashNAS strikes a middle ground, combining the leading industry-standard platform with purpose-built storage expertise. Integrating state-of-the-art NFS with Microsoft Windows Storage Server, FlashNAS provides industry-leading functionality and support as robust as that available from competitors such as EMC or NetApp—without needing a new OS or recreating Microsoft's entire CIFS protocol stack. FlashNAS uses innovative security-model mapping to seamlessly integrate NFS access permissions with Windows security to provide simple, transparent and secure data access.

That means lower costs, better reliability, and robust support—making Winchester Systems FlashNAS a natural choice as *the* low-cost, high-performance multiprotocol file server.

WINCHESTERSYSTEMS®
Purpose-Built Storage

Overview of FlashNAS Unified Storage

Winchester Systems has delivered the FlashNAS system to many clients over the years, primarily based on Windows' CIFS architecture. To accommodate the growing need for supporting both NFS and CIFS in one device, FlashNAS Unified Storage empowers IT to use one efficient, low-cost platform to meet the needs of the enterprise.

Some of the primary features of the FlashNAS Unified Storage include:

Software Feature	User Benefit
Windows-based platform	Industry-standard – Short learning curve
Configuration	Easy to use GUI
Feature-rich platform	De-dupe, snapshot and more . . .
CIFS and NFS protocols	Key standards including Windows, Linux,-Unix & MAC
Network protocol support	1GbE, 10GbE, 40GbE and InfiniBand
iSCSI support	Supports iSCSI SAN
Add-in remote replication	CA Replication for remote backup and disaster recovery
Hardware Feature	User Benefit
Dual RAID and NAS processors	Parallel processing for speed
Server-based NAS	Dedicated to network support
Hardware RAID	Dedicated to disk I/O
PCIe bus for expandability	Highly configurable
Serviceable and upgradeable	Hot-swap field replaceable units
Highly scalable	One server expands to 2PB
1U, 2U and 4U server options	The right appliance for each user's needs
1U	To 200 TB
2U	To 2 PB plus faster processors
4U	Redundant FlashServer-HA for high-availability
Host-based RAID and External RAID systems	Flexible options for meeting all needs
Host-based RAID	Low-cost, up to 200TB
External RAID	High speed RAID to 2PB in one cabinet

FlashNAS Unified Storage from Winchester Systems offers a very easy to use system that provides a myriad of options to meet the varying needs of clients, without sacrificing performance. This modular architecture allows for low-cost scalability, enabling low-cost upgrades throughout the life of the system.

The rest of this white paper explores multiprotocol file access challenges in depth, and explains Winchester Systems' approach to addressing those challenges with FlashNAS.

Why Multiprotocol NAS?

Believe it or not, there are IT shops—far and few between though they may be—that enjoy the luxury of a homogenous environment, running one operating system (OS) on all servers and desktops used by the organization. They share files over the network using a single, common protocol: typically either the Network File System (NFS) protocol used by most Unix and Linux systems, or Server Message Block (SMB),[i] which is predominantly used on Windows computers (CIFS protocol).

Most of us, however, must deal with a mix of OSes and network file-access protocols. The differences between the operating systems have been the subject of endless discussion and debate—sometimes approaching the intensity of religion—through the years. But these differences are also quite evident in their respective network file-access protocols. Each has a security model that reflects that of its native-OS file system. NFS, for example, uses a Unix file system (UFS) permissions model, granting *read, write* and/or *execute* access to three categories of user: the user itself, its associated group, and everyone else. SMB, of course, inherits its security model from the Windows NTFS file system, using Access Control Lists (ACLs) to explicitly grant or deny access to specific users or groups.

What's the easiest way to deal with this? Keep the two protocols and their data separated. And the simplest way to do that is to use separate file servers: Windows for CIFS and Unix/Linux for NFS. After all, general-purpose servers have been serving their files to other computers for almost as long as networks have existed. But when it comes to performance and scalability, it's no surprise that the race goes to storage-optimized platforms. More important, using general-purpose servers this way adds administrative complexity and cost. Routine functions such as backup and restore are duplicated. And it runs counter to the approach most organizations are taking to get more for their IT dollar: consolidating their physical servers and storage.

Ok, how about keeping NFS and CIFS protocols and their data separate, but using a common file server? The benefits, like those of a multi-function printer/scanner/copier/fax machine, are straightforward: removing cost and complexity, and increasing resource efficiency—without sacrificing functionality. And, in fact, this is the most likely scenario in today's IT shops.

So it's no surprise support for both NFS and CIFS has become a fundamentally required feature of network-attached storage (NAS) products. But there are times when applications running on Windows and Unix systems need shared access to the same data sets, regardless of the network protocol used. That means a NAS product serving those files must somehow provide consistent security for those files—regardless of the network protocol used.

Sounds reasonable. But there's a catch: because the ACL-based security model of NTFS is so much richer than that of UFS, a direct mapping between them is not mathematically possible. As a result, storage and software vendors have had to come up with non-mathematical strategies to blend these two models together as compatibly as possible.

The rest of this white paper describes the approach used by Winchester Systems to address this challenge, bringing industry-leading multiprotocol capability to its customers at a fraction of typical industry-leader prices.

Laying Groundwork

How does one go about building a multiprotocol NAS product that serves files to both Windows and Unix/Linux systems? The options can be summarized into three fundamental approaches:

> **Build on a Unix/Linux foundation.** This option begins with a Unix or Linux operating system and its built-in Unix file system and NFS stack, and adds CIFS-server software. The CIFS package is responsible for implementing the NTFS security scheme, and maintaining connection states. One popular example is Samba, an open-source package available for free. A frequent choice for do-it-yourselfers, this approach has attractively low acquisition costs, but comes with significant disadvantages.

The most obvious is obtaining support when problems occur. Less obvious is the challenge of ensuring interoperability with Microsoft's CIFS implementations in Windows, which tend to change—and even add features—with every OS update. Typically, Unix/Linux implementations rely on "lowest common denominator" security settings and Windows 2000 era credential exchanges.

Build on a Windows foundation. This option begins with a Windows Server platform, which provides built-in CIFS and an optional NFS stack. Storage vendors typically use a special OS version from Microsoft named "Windows Storage Server" that adds single-instance storage for greater efficiency, index-based search capabilities, and file-server performance optimization. This approach costs a bit more up front than a Linux-based platform, but has an obvious advantage: it's based on Microsoft's own implementation of its complex ACL-based security and CIFS protocol, ensuring interoperability—and support—regardless how complicated the network environment in which it's deployed. Microsoft's NFS stack provides basic functionality and simple integration. But it's chronically years behind in implementing current NFS versions, meaning NFS security and performance can lag behind the industry standard.

Build your own foundation. This option is the most ambitious, and is the one taken by NetApp and EMC for their NAS products. It requires the vendor to build its own operating system kernel and file system, as well as NFS and CIFS protocol stacks. And deal with bridging the two security models. NetApp chose a simple approach to handing user and permission mapping, which we'll describe later. EMC, on the other hand, offers several schemes for both. A storage admin must decide which user-mapping method to use before configuring file-sharing services, and which permissions-mapping method to use for each file system being served. Needless to say, both companies invested heavily in their implementations, and they charge accordingly.

Each choice seems to require an unfortunate trade-off between risk, capital cost and operating complexity. The least-expensive option carries the most risk and operational complexity, and the safest option is also the most expensive. Isn't there a middle ground somewhere? Can't someone find a way to combine the best of these options for customers at the lower cost *and* lower risk?

That's how the storage architects at Winchester Systems approached the problem. Leveraging over 25 years of NFS expertise, the company integrated its own NFS server stack with Windows Storage Server 2008 R2. The result: a low-cost, high-performance file server that provides current, state-of-the-art features and stability for both NFS and CIFS protocols.

It may not seem obvious, but the rich NTFS file-system semantics and security model form an excellent platform for building a full-featured NFS server. Now that Winchester Systems engineers had a solid foundation, they needed to deal with two dramatically different security schemes.

A Tale of Two Models

Sun Microsystems, an early pioneer in Unix and TCP/IP networking, developed NFS in the mid 1980s as a way to enable users to access files on a remote machine in the same way as a local file. Sun's top design goals for NFS were simplicity, performance, and multi-vendor interoperability, and NFS gained widespread use on Unix systems during the 1990s.[ii] After some setup by an administrator, remote files appear within the Unix file system directory tree and are indistinguishable from files on a directly attached disk drive. The NFS protocol itself is widely described as "stateless," meaning each exchange of messages over the network has no dependence on previous exchanges. Though this approach has interoperability benefits, and makes transparently riding out network link failures simpler, it also makes it very difficult to add more-advanced file system capabilities such as locking.

Naturally, NFS uses the same security model as UFS.[iii] Files and directories are owned by a user, and are also assigned to a group (usually the same group to which the file owner belongs). Access is controlled for each of three categories: *user* (the owner)*, group,*

and *other.* A user that is neither the file owner nor a member of the file's assigned group is considered "other." There are three types of access permission specified for each category:

- *Read*: allows reading file contents. When set for a directory, allows reading the list of file names within that directory. Accessing those files is not allowed unless the execute permission is also set for the directory (and each file's permissions allow it).
- *Write*: allows writing file contents. When set for a directory, allows creating, deleting, and renaming files within that directory—if execute permission is also set for the directory.[iv]
- *Execute*: allows executing a file as a program, including application binaries and shell scripts. When set for a directory, allows accessing a file within the directory if its name is known (and that file's permissions allow it).

Creating, deleting and renaming files, as well as changing attributes such as owner or permissions, works the same way over NFS as it does on a local file system. That means individual users can change file and directory permissions on their files whenever they wish to change what they share, and how widely they share it.

The evolution of CIFS was similar, and started around the same time. It was originally designed by IBM to extend the local DOS file system into a networked file system. Microsoft later modified it extensively to form a networked extension of the Windows NT file system (NTFS).[v] After some setup work by an administrator, a new "drive" appears on the Windows system, and behaves as if directly attached to the system. To pull this off, CIFS maintains state information about connections to the server, and even connections to individual files for coordinating simultaneous shared access and caching across multiple client systems.

Naturally, CIFS uses the NTFS security model. Instead of a permission attribute, like in Unix, each NTFS file or directory has a security descriptor that contains, among other things, the owner's Security ID (SID) and an Access Control List (ACL).[vi] Each entry in the list, called an Access Control Entry (ACE), explicitly allows or denies access to a single user or group. The "standard permissions" that can be specified include:

- *Full Control*
- *Modify*
- *Read*
- *Read & Execute*
- *Write*
- *List Folder Contents (folders only)*

These standard permissions are actually groupings of "Special Permissions," which are more detailed—and more numerous. For example, the standard *Read* permission includes special permissions *List Folder/Read Data, Read Attributes, Read Extended Attributes, Read Permissions,* and *Synchronize*.[vii]

When a user requests access to a file, each ACE in the file's ACL is checked—in order of their appearance in the list. When an ACE is found that matches the user/group and the requested access, the search stops and the request is either granted or denied. If no matching ACE is found, the access is denied. This provides an incredibly flexible means to enforce complex security rules such as "allow read and write access by all users in one group unless they are also members of another group, in which case they get read-only access."

The differences between NFS and CIFS security models are stark. Translating a set of permissions from one model to the other is a daunting challenge at best. Yet the need for applications to shared access to networked file data, regardless what operating system they run on, is real. So Winchester Systems needed to come up with a way to merge these two models.

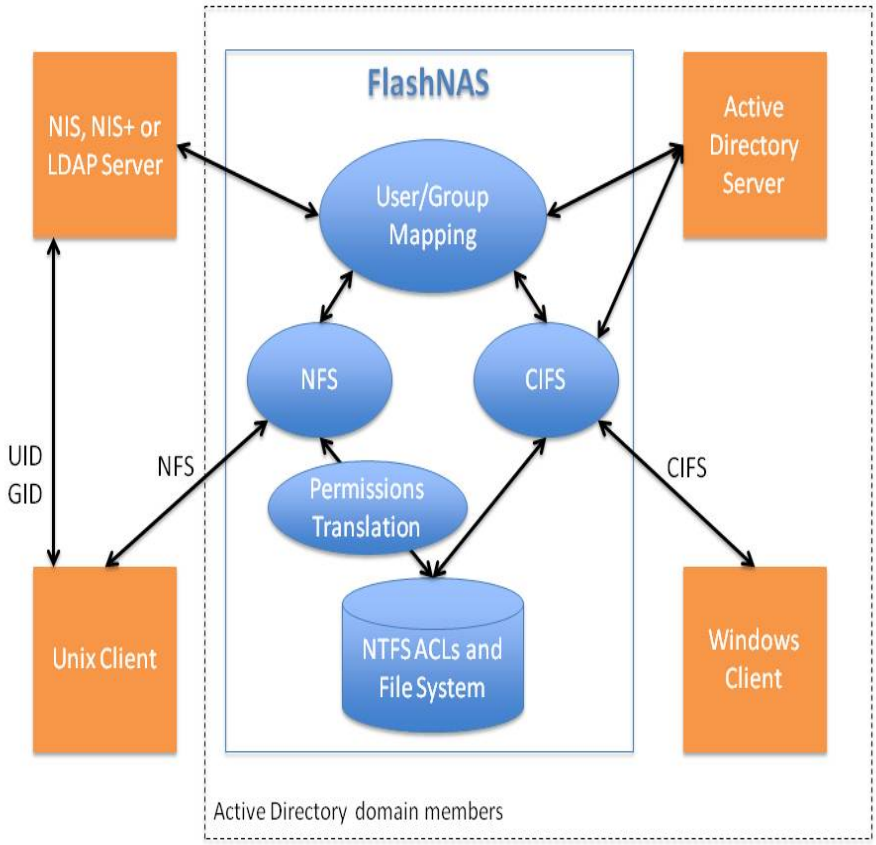

Figure 1, FlashNAS authentication and account mapping

It Starts With A User

The good news is that the security models in both Unix and Windows, and thus NFS and CIFS, are based on the concept of authenticating a user, and checking whether that user is authorized to access an object. That means mapping Unix users to Windows users, and vice-versa, can form the basis for a bridge between the two models.

For Windows-based users accessing FlashNAS via CIFS, the process is obviously transparent. Because it's based on Windows Storage Server, FlashNAS can participate seamlessly in one or more Windows domains. For Unix-based NFS users, Winchester Systems makes this process equally transparent. Unix NFS requests include User Identifiers (UIDs) and Group Identifiers (GIDs), which are mapped to a corresponding Windows-based network user, and file access is provided in the context of the mapped user to ensure authenticated access.

Access may be provided to both local users and domain users.

In addition to one-to-one mapping between Windows and Unix user and group accounts, FlashNAS permits one-to-many mapping. This lets you associate multiple Unix users with a single Windows user, or multiple Windows users with a single Unix user. This can be useful, for example, when you do not need to maintain separate Unix accounts for individuals and would rather use a few accounts to provide different classes of access permission.

FlashNAS provides two name-mapping methods: automatic and manual. Automatic mapping takes NFS user and group names and matches them to their corresponding Windows user and group names. This automation reduces the risk of incorrect mapping—and saves administrators a lot of time creating and maintaining maps compared to entering user and group mapping one at a time. Administrators can select all Windows users, specific domains, or even specific users per domain for automatic mapping. Of course, mapping entries can also be created and modified manually, and can be used in conjunction with automatic mapping.

For users and groups that do not appear in the mapping database, an administrator can specify "default mappings." A common example would be to map user "nobody@abc.com" to "\\WIN_DOMAIN\Guest" and group "nobody@abc.com" to "\\WIN_DOMAIN\Guests." Unix and Windows use those names to indicate an unauthenticated user, which by default has severely limited access (if any). Using this default mapping ensures an unauthenticated user cannot be inadvertently mapped to a real user.

FlashNAS also periodically refreshes its mapping database from the source databases, ensuring that it is always kept up to date as changes occur in Windows and UNIX name spaces. You can also refresh the database anytime you know the source databases have changed. By default, the name mappings database is stored on the local system. An administrator can also configure name mappings to be stored on a network-wide Active Directory server via the Lightweight Directory Access Protocol (LDAP), providing a centralized database for use by multiple FlashNAS servers. If desired, administrators can also export or import mappings to or from a disk file.

FlashNAS can obtain Unix user and group information from a directory service such as NIS, NIS+, and LDAP, or directly from a Unix host,[viii] and supports user authentication performed by NFS clients (the default for most Unix systems) and Kerberos.

Ok, so FlashNAS has a straightforward way to map Unix NFS users to Windows users. Now comes the tricky part. Translating file and directory permissions!

ACEs Up Their Sleeve

Because of the richness of Windows' ACL-based security model, FlashNAS stores all of its permissions information—regardless of what type of user set them—as ACLs in the NTFS file system. The permissions that apply to a given NFS user depend on the Windows user and group names to which that user is mapped—and if that user has an "Access-Allowed" Access Control Entry (ACE) defined in the ACL for the resource requested.

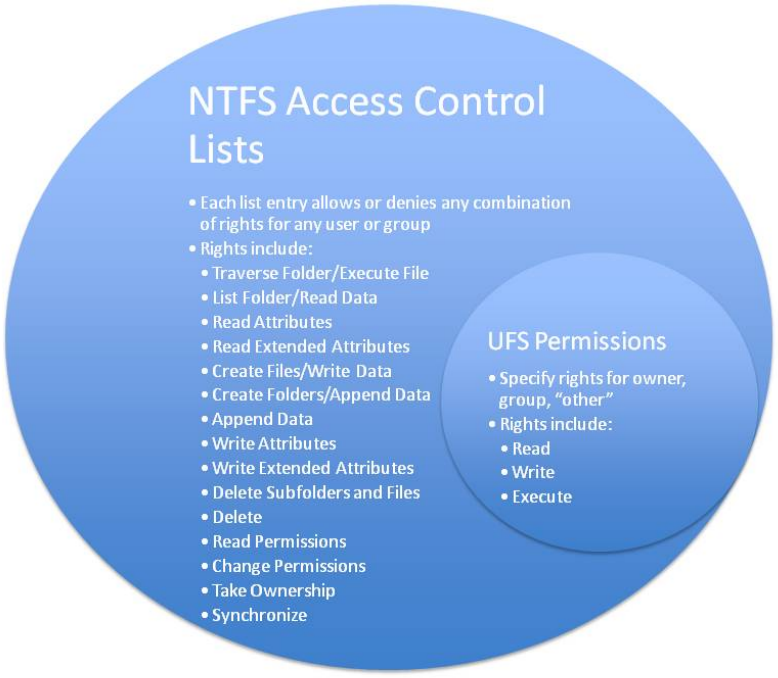

Figure 2, Unix File System security is a subset of NTFS ACL-based security

The reason FlashNAS can do this is because Unix File System security is, with a couple minor exceptions,[ix] a subset of the rich NTFS security model (see figure 2).

When an NFS client accesses an existing file or directory, FlashNAS determines the permissions for that client by combining every ACE that applies to the client based on the name mappings. Following standard Windows ACL-handling rules, FlashNAS processes ACEs in the order in which they appear in the ACL. Because multiple ACEs may apply to a given NFS user, and ACEs can either grant or deny permissions, the combined set of entries may result in multiple ACEs applying to the same user for a given permission. When this happens, FlashNAS applies the first entry found in the ACL.

After consulting the ACL for a resource, FlashNAS grants a given NFS user access to the resource if the following conditions are true:

- The NFS user is mapped to a user or group that has an Access-Allowed ACE in the ACL, or the Everyone group has an Access-Allowed ACE in the ACL.

- There are no Access-Denied ACEs earlier in the ACL for the user and groups (including Everyone) that apply to the NFS user.

When an NFS client creates file or directory, or requests a change in permissions, FlashNAS sets the Access Control List (ACL) for the affected resource according to the permissions set by the client. Because UNIX permissions are not identical to NTFS permissions, FlashNAS translates the permissions in the following way:

- For each "User" permission set by the client, FlashNAS adds the corresponding NTFS permission to the Access-Allowed ACE for the Windows user that owns the resource.
- For each "Group" permission set by the client, FlashNAS adds the corresponding NTFS permission to the Access-Allowed ACE for the Windows group that owns the resource.
- For each "Other" permission set by the client, FlashNAS adds the corresponding NTFS permission to the Access-Allowed ACE for the Windows group Everyone.[x]

Because it directly creates and modifies ACEs in a resource's ACL, FlashNAS can create ACLs with ACEs arranged in an order that wouldn't normally appear on a Windows system—but are needed to get the desired resulting permissions requested by an NFS client.

For example, suppose Unix user "joe" wants to share a program file so that all users have full access, except members of his primary group, "sales," to which he wants to grant read-only access—and he wants to guarantee he still has full access. The Unix permissions he'd want to set are:

- User: read, write, execute
- Group: read, execute
- Other: read, write, execute

The resulting ACL needs to grant full access to everyone except that group. Windows standard canonical ACL rules require Access-Denied ACEs to be placed before Access-Allowed ACEs.[xi] But if the ACL was to have an Access-Denied ACE for write access by group "sales" before the Access-Allowed ACE for full control by user "joe," that user would end up being denied write access because "joe" is a member of the "sales" group. So FlashNAS writes the ACEs in a different order, granting "joe" full access before denying write access to "sales."[xii]

If a CIFS user later changes the permissions, the ACL will be overwritten as requested by the Windows user—following the Windows canonical rules. And that's fine; a Windows user expects Windows ACL behavior. Just as an NFS user expects NFS permissions behavior. In other words, FlashNAS assigns permissions to best enforce the rules appropriate for the authorized user—and platform—that set them.

Other multiprotocol NAS vendors offer similar permissions-handling behavior. For example, Netapp uses its own operating system and file system so-stored files can have either Unix permissions or a Windows ACL—but not both. If a file has a Windows ACL and a Unix user changes permissions via NFS, the ACL is deleted and Unix permissions are created. Conversely, if a file has Unix permissions and a Windows user goes to change them, the Unix permissions are deleted and an ACL is created.

EMC, which also uses its own file system in its Celerra product, can also write either Unix permissions or Windows ACLs. In fact, it always writes both. The company also requires administrators to choose from a half dozen access-checking policies for each file system mounted on the NAS server. Policy options include maintaining both Windows ACLs and Unix permissions but enforcing them only on their respective protocols, maintaining both sets of permissions and allowing access from either platform only if both sets of permissions allow it, maintaining both sets of permissions and modifying both regardless which protocol set it, maintaining

both sets but only checking the one that was last written, and so on. It's no surprise, then, to find EMC's multiprotocol guide alone is around 80 pages long!

Keeping It Simple

Most organizations are already dealing with more than enough complexity in their IT shops. And operational costs, which are usually mostly human labor, still dominate most IT budgets. Sharing files among applications, regardless what operating system they run on, needs to be as simple—and efficient—as possible.

For most IT shops, that means consolidating file shares onto Network Attached Storage servers that can be accessed from any operating system using its native file-sharing protocol. Multiprotocol support has become a critical feature of leading NAS vendors. But the way NAS servers handle the idiosyncrasies of Unix and Windows security, and their respective network file access protocols NFS and CIFS, is also a huge factor in keeping IT costs under control. The simpler and more efficient, the better!

By integrating state-of-the-art NFS with industry-standard Windows Storage Server, Winchester Systems brings industry-leading functionality and support to FlashNAS—as robust as that available from competitors such as EMC or NetApp—without needing to implement its own OS kernel and file system, recreate Microsoft's entire CIFS protocol stack and Active Directory integration, or cobble together open-source packages.

That means lower costs, better reliability, and robust support—making Winchester Systems FlashNAS a natural choice as *the* low-cost, high-performance multiprotocol file server.

Footnotes

[i] SMB is also often referred to as Common Internet File System (CIFS). CIFS was originally a proposed standard from Microsoft based on Windows NT 4.0 and Windows 2000 implementations of CIFS. However, that standard was not adopted by the industry. Microsoft later redefined CIFS to refer to a now-obsolete Windows NT LAN Manager (NTLM) implementation of SMB. Despite this interesting bit of trivia, most IT people use the terms CIFS and SMB interchangeably.

[ii] The first widely adopted version was NFS version 2, published as a standard by the Internet Engineering Task Force in 1989 (see http://tools.ietf.org/html/rfc1094). NFSv3, which added new capabilities such as larger file support, was published in 1995 (see http://www.ietf.org/rfc/rfc1813.txt).

[iii] NFS version 4, which has steadily displaced version 3 on deployed Unix and Linux systems, improves client-server protocol security and adds Access Control Lists (ACLs). FlashNAS supports NFSv4 and its associated ACLs. However, most Unix/Linux administrators view ACLs as cumbersome and continue to use only basic Unix File System (UFS) permissions. So for simplicity, this paper focuses on the more limited NFSv3 security model. For detailed information on FlashNAS NFSv4 features, refer to the product documentation.

[iv] Without execute permission, write permission on a directory is effectively meaningless. What's more, if a user has write and execute permission to a directory, they can delete and rename files within that directory *regardless of permissions set on the individual files*. This is an often-misunderstood feature of Unix directories.

[v] The company introduced a significantly new version, SMB 2.0, with Windows Vista; SMB 2.1 with Windows 7 and Windows Server 2008 R2; and SMB 3.0 with Windows 8 and Windows Server 2012. Although proprietary, Microsoft has published the protocol specification for other vendors to build interoperable products (see http://msdn.microsoft.com/en-us/library/cc246482(PROT.13).aspx).

[vi] There are actually two access control lists in NTFS: the Discretionary Access Control List (DACL), and the System Access Control List (SACL). The DACL controls access to an object, and the SACL controls logging of access attempts. ACLs referred to in this paper are DACLs.

[vii] For a more detailed discussion of NTFS access control and permissions, see http://technet.microsoft.com/en-us/library/cc770749.

[viii] A Unix host stores its local User ID (UID) and Group ID (GID) data in two files: a *passwd* file that associates each user with a password, a UID and a GID; and a *group* file that associates each group with its GIG and the UIDs of the group's members. Obtaining this information from a Unix host involves copying these two files to the Winchester Systems NFS Server system.

[ix] The "sticky bit," for example. Refer to your favorite Unix documentation for more information.

[x] FlashNAS maps the Unix permission "Other" to the special Windows group Everyone. However, setting permissions for the Everyone group in Windows is not the same as setting "Other" permissions in Unix. All users and groups in Windows belong to the Everyone group, including the user and group owners of shared resources. "Other" permissions for a resource apply only to Unix users who do not own the resource and do not belong to the primary group of the owner. Therefore, when a user or administrator sets "Other" permissions for a shared resource, you are actually setting permissions for all users and groups that can access that resource.

[xi] The full canonical rules include handling of ACEs inherited from a resource's parent, grandparent, and so on up the directory tree.

[xii] Yes, the desired behavior can be obtained by writing a different ACL that can adhere to Microsoft's canonical rules, but doing so in the context of programmatically translating Unix permissions to Windows ACLs—and back again—is impractical.