

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

NETSKOPE, INC.,

Petitioner,

v.

FORTINET, INC.,

Patent Owner.

PTAB Case No. IPR2023-00456

Patent No. 11,036,856 B2

**REBUTTAL DECLARATION OF DR. MICHAEL FRANZ IN SUPPORT OF
PETITIONER'S REPLY TO PATENT OWNER'S RESPONSE**

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	RESPONSES TO PATENT OWNER RESPONSE	2
A.	Obviousness of the “copying” limitation in the ’856 claims has been established.....	2
B.	The references relied upon by Petitioner would have been obvious to combine	17
C.	In claims 8/19/27, the location of the second repository is merely an implementation decision	22
D.	The Sharpe-Seese-Kamalapuram-Dods combination rendered claims 3, 6, 11, 15, 17, 23, and 25 obvious.....	24
E.	PO repeatedly misquoted or mischaracterized my testimony	24

LIST OF APPENDICES

Appendix G	Chris Hoffmann, <i>What is ReFS (the Resilient File System) on Windows?</i> June 26, 2017.
Appendix H	Andrew Cunningham and Lee Hutchinson, <i>macOS 10.12 Sierra: The Ars Technica Review</i> , September 20, 2016.

I. INTRODUCTION

1. I, Dr. Michael Franz, have been retained by Petitioner Netskope, Inc. (“Petitioner”) to investigate and opine on certain issues relating to United States Patent No. 11,036,856 (“the ’856 patent”). Petitioner requests that the Patent Trial and Appeal Board (“PTAB” or “Board”) review and cancel claims 1-28 of the ’856 patent.

2. Last year, I provided a declaration in support of Petitioner’s IPR Petition. My Opening Declaration is Exhibit 1002 to the Petition and provides an explanation of my qualifications, a discussion of the technology relevant to the ’856 patent, and my opinions with respect to the ’856 patent.

3. I have prepared this Rebuttal Declaration to address arguments made in the Patent Owner Response (“POR”) and the accompanying Declaration of John Black in support of the Patent Owner Response (Ex. 2004, “Black Response Declaration”).

4. In addition to the materials referenced and cited in my Opening Declaration, I have now reviewed and considered the Board’s Institution Decision, Patent Owner’s Preliminary Response and its Response, the declaration of John Black in support of Patent Owner’s Preliminary Response, Dr. Black’s declaration in support of Patent Owner’s Response, and the transcript of the deposition of Dr. Black.

II. RESPONSES TO PATENT OWNER RESPONSE

A. Obviousness of the “copying” limitation in the ’856 claims has been established

1. PO interprets a “copying” operation more narrowly than would a POSITA

5. It is my understanding that Dr. Black contends “[c]opying” information requires creating, at least temporarily, a second copy of portions of that information and “[m]oving” merely requires changing the location of the information within a file, and that the ’856 patent does not use those terms interchangeably. (Black Supp. Declaration, ¶¶48-61.) Dr. Black contends “moving” occurs when data is moved within the same file share (what he calls “moving without copying” or Moving Type I), while “copying” occurs when data is moved between file shares (what he calls “moving with copying” or Moving Type II). (Black Dep. Tr., 9:5-10:11.) Dr. Black also contends “plain old copying,” “sharing,” and “transferring” are not interchangeably used in the ’856 patent and the industry. For the reasons below, I disagree.

a. The ’856 patent uses “moving” and “copying” interchangeably

6. Various sections of the ’856 patent use moving/copying interchangeably. Figures 1a and 1b speak of “moving” files and do not mention copying at all. The specification additionally uses the terms “transfer” (8:28) and “copying” in (8:56) to refer to those same “moving” operations:

“mak[ing] the clean file accessible to the users by **copying and/or transferring** the clean file from dirty file store 106 to clean file store 110 that is accessible to the users.”

7. So the specification uses “moving,” copying,” and “transferring” to refer to the same operations. While the specification uses the terminology “copying and/or transferring,” the almost exact same language is re-iterated in the claims as:

- makes the clean file accessible to the users by, [...], copying the clean file from the first repository to a second repository that is accessible to the users (Claim 1)
- making the clean file accessible to the users by, [..], copying the clean file from the first repository to a second repository that is accessible to the users (Claim 13).
- make the clean file accessible to the users by, [...], copying the clean file from the first repository to a second repository that is accessible to the users (Claim 21).

8. Claim 12 recites both “moving” and “copying” and uses the terms interchangeably. Claim 12 depends from claim 1, which, as noted above, states “copying the clean file from the first repository to a second repository.” Claim 12 recites:

12. The secure data transfer system of claim 1, wherein the network security device makes said determination regarding whether the untrusted file is a clean file that is free of malicious content by first moving the untrusted file from the first repository to a third repository, and then applying one or more security checks to the untrusted file in the third repository, and moving back the checked file as a clean file into the first repository for onward copying to the second repository or moving the checked file as a clean file directly from the third repository to the second repository.

(’856 patent, cl. 12 (highlighting added); *see also* ’856 patent, 6:36-45.) Here, the terms “moving” and “copying” are used interchangeably. Under PO’s view of “copying” and “moving,” claim 12 makes no sense. If “copying” is only used when moving files to a different file system, then the first repository and second repository would need to be separate file systems. But that cannot be true because the third repository “moves” files—an operation PO contends only happens when files are moved between the same file system—to the physically-separate first and second repository

9. In the latter option where the file is moved from the first repository to the second repository, the operation of “copying” a clean file from the first repository to the second repository (from claim 1) is realized by “moving” the file to the third repository then “moving” the file again to the second repository – thus appearing to perform a “copy” operation by two “move” operations.

10. Other parts of the '856 patent also use “moving” and “copying” interchangeably with other operations including “transferring” and “sharing,” as the Board noted. (ID, 22-23.) For example:

- Claim 5: “*copies* the file to the second repository by *sharing* it by means of any or a combination of network file system (NFS), file transfer protocol (FTP), common Internet file system (CIFS), Internet Small Computer Systems Interface (iSCSI), Storage Area Network (SAN), and local storage.”
- Figure 4A, reference 408: “*moving* clean files into clean S3 bucket.”
- 6:5-6: “In an embodiment, the network security device can *copy* the file to the second repository by *sharing* it.”
- 8:54-57: “network security device 108 can make the clean file accessible to the users by *copying and/or transferring* the clean file from dirty file store 106 to clean file store 110 that is accessible to the users.”
- 10:17-22: “clean file transfer module 204 can make the clean file ... accessible to the users by, *copying or transferring* the clean file from a first repository (e.g., dirty file store 104) that is inaccessible to end users to a second repository (e.g., clean file store 110) that is accessible to end users.”

- 10:28-29: “clean file transfer module 204 can copy the file to the second repository by sharing it.”

11. Finally, to support their interpretation of the claimed “copying” operation, PO points to Figure 1B and argues that the patent claims “copying” because “‘moving’ files without ‘copying’ is not practical” in the Figure 1B implementation. (POR, 20, 22.) “Circumstances” envisioned by the patent allegedly motivate a POSITA to “to copy files rather than move them without copying.” (Black Supp. Decl., ¶60.)

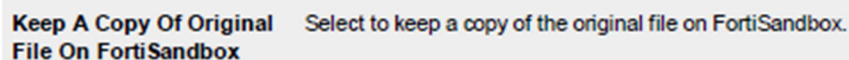
12. I disagree with these arguments. They result in PO reading out embodiments of the ’856 patent. First, both Figure 1B *and* Figure 1A state that “Clean Files are *Moved* to Clean File Store” (emphasis added). There is no reason to conclude either from the figures or from the corresponding description in the specification that “moved” in one figure has a different meaning than “moved” in the other figure. I also note that the fact that Figure 1A uses “moved” and “copied” when describing a file going from the first to second repository on the same data store undermines PO’s argument that “copying” is only used when data moves to a different file store.

13. Furthermore, even if a POSITA would understand that “moving” a file can be implemented in different ways depending on the configuration of the file system (*see infra*, ¶¶23-25), claim 1 does not specify the location of the second

repository and thus is not limited to the particular implementation in which “copying” means “moving by copying” that PO apparently believes to follow from Figure 1B. Dependent claim 9 in fact states that the second repository is “part of the public cloud file store,” like the first repository, and thus is directed to the embodiment of Figure 1A. The specification states that the “sanitized storage area...may or may not be a part of the public cloud file store.” (’856, 3:34-35.) PO’s interpretation of “copy” as an operation that is performed when the first and second repositories are located on separate storage volumes reads out the dependent claims and the embodiments in the specification that do not require this configuration.

b. PO’s own references use “moving” and “copying” interchangeably.

14. PO’s attempts to distinguish between “moving” and “copying” are undercut by their own literature. First, the FortiSandbox manual cited in the Petition, provides an option to “keep a copy of the original file on FortiSandbox.” See the following excerpt from p. 115:

A screenshot of text from the FortiSandbox manual. The text is displayed in two columns. The left column contains the text "Keep A Copy Of Original File On FortiSandbox" in bold black font. The right column contains the text "Select to keep a copy of the original file on FortiSandbox." in a regular black font. The background is a light gray color.

Keep A Copy Of Original File On FortiSandbox Select to keep a copy of the original file on FortiSandbox.

(FortiSandbox, p. 115)

If this option is one that the user can select, whether a copy remains is immaterial to the operation of the FortiSandbox device.

15. Additionally, Ex. 2003, a later version of the FortiSandbox Administration guide, uses “moving” and “copying” interchangeably. Although Ex. 2003 was published in 2023, its use of “moving” and “copying” reflects what a POSITA would have understood in 2018, at the time of the ’856 patent’s filing, because there were no significant changes to these fundamental file system concepts between 2018 and 2023. For example, the following excerpt from p. 76 uses the combined term “copying or moving”.

Enable <u>copying or moving</u> clean files to a sanitized location	<u>Copy or move files</u> with a Clean rating to another location. By default, a new folder is created for each scheduled scan job in the sanitized location and all clean files are <u>copied into it</u> with the original folder structure. To save space, uncheck <i>Keep a complete copy of clean files for every scheduled scan</i> so that files of the same path have only one copy in the sanitized location.
--	---

(Ex. 2003, p. 76; annotated)

16. Even more confusingly, or perhaps amusingly, the later edition of the FortiSandbox manual provided in Ex. 2003 uses the terminology of “moving” in conjunction with the option of “leaving a copy in its original location.” In the excerpts below, what the PO now calls “copying” appears to have been described as “moving while keeping a copy.”



When a file is moved, to leave a copy in its original location, go to the Quarantine edit page or sanitized share and select the *Keep Original File At Current Location*.

(Ex. 2003, p. 76; annotated)

Keep Original File At Current Location	Select to keep the original file at the current location when a file is quarantined from a network share. <u>By default, the original file is kept at its current location when being moved.</u>
---	--

(Ex. 2003, p. 79; annotated)

Enable moving clean files to a sanitized location

Select to move Clean rating files to another location. By default, a new folder is created for each scheduled scan job in the sanitized location and all clean files are copied under it with the original folder structure. To save storage size, the user can un-check *Keep a complete copy of clean files for every scheduled scan*, then files of the same path will have only one copy saved in the sanitized location.

(Ex. 2003, p. 99; annotated)

c. PO's distinctions between “moving” and “copying” are inconsistent with the way a POSITA would understand these terms

17. The PO keeps making the argument that copying files is “less efficient” than moving files because “the amount of processing required to copy a real-word [sic] file is more than the amount of processing required to modify file system directory entries.” (POR, 22.)

18. This is a naïve understanding of how modern file systems actually work. A POSITA having at least two years of practical experience with computer networking and computer security (Franz Op., ¶28) would have had a much more advanced understanding of modern file systems and their operations, based on actual systems in use at the time and their specific deployment to counter computer security threats. In particular, a POSITA would be familiar with “copy-on-write functionality” introduced into modern file systems from 2008 onwards.

19. In such copy-on-write file systems, additional disk storage blocks are allocated only at the time that one of the copies is modified (i.e., “upon writing”), to store the modified bits of the affected copy—while any remaining copies continue to share the original bits.

20. Because of its ability to create almost instantaneous copies of files without having to duplicate the underlying bits, copy-on-write file systems rapidly gained in popularity. For example, the copy-on-write feature is supported in Microsoft’s Windows Server 2012 operating system, released in August 2012¹, and in Apple’s APFS File system, announced in June 2016 and rolled out to both iOS and MacOS in the following year.²

21. The functionality of copy-on-write has been widely used, and would have been known to a POSITA, for applications that include combating the problem of ransomware, which is malware that surreptitiously encrypts an organization’s files and will release the encryption key only upon payment of a ransom payment.

22. A common defense against ransomware is taking “snapshots” of the file system, each of which is typically an inaccessible hidden copy of the entire file system’s contents. Obviously, it would be extremely wasteful to duplicate the bit-

¹ See Appendix G

² See Appendix H

level content of every file for every snapshot. This is where “copy on write” comes in: in a file system providing copy-on-write, “copying a file” entails only a directory pointer operation if both the source and destination are mapped to the same storage volume: a duplicate directory entry is created that points to the same sequence of disk blocks as the original file. Hence, one can create as many copies of a file as one likes without requiring any additional storage space beyond the extra directory entries, and the cost of the “copy” operation itself is independent of the length of the file.

23. Hence, as I explained previously, a POSITA would understand that in a modern operating system, “moving” and “copying” of a file on the same storage volume are both mere directory entry operations that require no file content bits to be duplicated, while “moving” and “copying” of a file to a different storage volume both require the actual duplication of content bits. Hence, the “efficiency” of such an operation is not determined by whether it is “copying” or “moving”, but whether source and destination are on the same volume or not. The following table summarizes this:

	destination path is mapped to a different storage volume as the source	destination path is mapped to the same storage volume as the source
--	--	---

file-level “copy” operation	(1) the blocks of the file are duplicated to the destination medium (effort proportional to the size of the file), then a new directory entry is created to point to the newly allocated storage blocks	(2) a new directory entry is created that points to the existing storage blocks of the original file
file-level “move” operation	(3) the blocks of the file are duplicated to the destination medium (effort proportional to the size of the file), then the original directory entry is modified to point to the newly allocated storage blocks	(4) the original directory entry is modified and continues pointing to the existing storage blocks of the original file

24. It is also worthwhile to note that in (3) above (moving a file from one storage volume to another), the original bits are usually not actually deleted from the storage volume until the corresponding blocks are re-used for storing something else. So, at least temporarily, a second copy of the file contents exists.

25. In summary, whether or not content bits are duplicated depends on the configuration of the filesystem and not on the operation used: If source and

destination are located on different volumes, both “copying” and “moving” would entail duplication of bits. If the source and destination are on the same volume, “copying,” like “moving,” can be implemented by just updating a pointer in a file system directory without duplication of bits (at least until one of the copies is modified).

26. A POSITA would have been able to determine how exactly to implement either of these operations within a system, based on the particular configuration of the system.

d. To achieve the expressed purpose of the alleged invention, it is immaterial whether one is “moving” or “copying”

27. The relevant purpose of “moving”/ “copying” in the alleged invention described in the ’856 patent is for “making clean files available to the end users” after they have successfully passed the determination phase. For this goal, it makes no difference whether they are moved or copied, i.e., it is immaterial whether possibly inaccessible duplicate copies of the clean files remain in the dirty file storage or not.

28. One could go even further and argue that *requiring* that such inaccessible duplicate copies remain in the dirty file storage is nonsense. It most certainly isn’t an “invention.”

e. PO's definition of "copying" makes their argument nearly impossible to follow

29. PO's convoluted distinctions between "copying," "moving by copying," "moving without copying," etc., make it challenging to understand both the scope of the '856 patent and their arguments in this proceeding. For example, as discussed above, Figures 1A and 1B both state that files are "moved" to a clean file repository and, by PO's argument, one is supposed to be able to conclude from Figure 1B that its moving operation includes a copy operation. At the same time, PO argues that "moving by copying" would not have been obvious when Petitioner's references use the term "moving." It is entirely unclear how one would know that "moving" includes "copying" in one instance, but would not know this in another instance.

30. In another example, Dr. Black distinguishes between a "type 2 move, which is copy and erase" and "a plain old copy." (Black Dep. Tr., 18:5-6.) Dr. Black explained how "moving by copying" differs from "just copying" as follows: "[M]oving by copying implies that you're doing a copy and then an erasure of the original....Copying implies that you make a copy and you don't remove the original." (*Id.*, 27:11-19.) Where claim 1 recites "copying," then, how would a POSITA know what the scope of this operation would entail? The internal contradictions of PO's argument is brought even more into focus in the context of claim 12, which recites multiple "moving" operations and a "copying" operation.

2. Burdett renders obvious “copying” a file

31. Burdett describes “moving” a file and describes one example in which this is performed by changing a path in a file system. (Burdett, 25:28-31.) This is consistent with “copying” a file, at least in some cases. For example, as described above, “copying” can be implemented by updating a pointer if the source and destination are on the same volume using the well-known copy-on-write technique. A POSITA would have known that using this technique would in fact have the same performance characteristics as “move”.

32. Even implementing a “copy” operation by other techniques would not materially impact efficiency of Burdett’s system. PO’s only argument against the efficiency of a “copy” operation relative to a “move” operation compares the processing requirements of a “move” operation within the *same file system on the same volume* to a “copy” operation between *different file systems or volumes*. (POR, 2, 24-27.) This comparison is irrelevant. For example, a “move” operation from one volume to another would entail similar duplication of bits as a “copy” operation because – even as admitted by PO – the only material difference between these operations is whether an original file is deleted from its initial location. (Black Dep. Tr., 10:10-11.) Besides, the performance impact of making even full copies of all the content bits of a file would likely be immaterial in comparison to other, far more

expensive operations in Burdett’s overall system, such as the cost of performing the security checks on the files.

3. The Sharpe-Seese-Kamalapuram combination renders obvious “copying” a file

33. As described in the Petition, Kamalapuram discloses or renders obvious “copying” a file to a second repository. (Pet., 67-68.) Kamalapuram describes “moving” when it stated, “a file may be stored in an isolated area of the network storage system while waiting to be scanned, and may be *moved* to a general storage area only after the file has been scanned by a security analysis system of the network storage system.” (Kamalapuram, [0122].) Kamalapuram does not specify whether the “isolated area of the network storage system” and the “general storage area” are located on the same physical volume. A POSITA would have known how to implement a “move” under various configurations of the file system as described above, including how to implement “moving by copying.” (See *infra* at ¶¶23-26.)

34. Likewise, for the same reasons as those discussed with respect to Burdett, implementing a “copy” operation in the combined Sharpe-Seese-Kamalapuram system would not materially impact efficiency. Recycling their arguments against Burdett, PO argues that “it would be nonsensical to implement Kamalapuram’s ‘moving’ using any copying – the much more efficient solution would be to ‘move’ files by changing file system directory entries and thus *without* ‘copying.’” (POR, 32.) But once again, this argument (1) assumes that the ’856

claims *only* cover the implementation in which the first and second repositories are located on different volumes, (2) reads into Kamalapuram a non-existent requirement to *only* move files within the same file system, and (3) makes unreasonable efficiency comparisons accounting for only some, but not all embodiments of the systems in question.

B. The references relied upon by Petitioner would have been obvious to combine

1. Burdett and FortiSandbox would have been obvious to combine

35. PO argued that the Burdett-FortiSandbox combination would not have been obvious because “moving Burdett’s scanning service 304 from its original location in cloud infrastructure 302 to the FortiSandbox device would be *detrimental* to the performance of Burdett’s system.” (POR, 28.)

36. I disagree. Burdett describes copying a file to repositories outside the cloud file store for malware scans. For example, Burdett describes how a file is “copied to the data store of the scanning service” to be scanned for malicious content. (Burdett, 29:34-38.) Later, Burdett similarly states that “The scanning service may copy the file from the cloud data store to another data store” for scanning the file, where “[t]he another data store may be a local data store.” (Burdett, 30:59-62.) Based on this disclosure, there would not be significant inefficiencies caused by

copying a file to a local data store on the FortiSandbox device for scanning by the FortiSandbox device.

37. Furthermore, performing scans on a device on an enterprise network would also have been obvious because of concerns such as privacy and controllability, as further explained in the Petition and my supporting declaration. (Pet., 19; Ex. 1002, ¶¶43, 46, 69-70, 88-89.) In an example, an enterprise may, as they often do, want to control their scans, such as wanting the ability to perform bulk scans, scheduling scans, or performing on-demand scans when they want, or performing more advanced security scans. Using a cloud service provider's scanning can hamper that control because the cloud service provider controls which types of scans are available, when and how they can be done, the uptime of the service, the security of the service, etc.

38. Dr. Black identifies several purported disadvantages to the combination of Burdett and FortiSandbox. I disagree with each of these.

Negative Repercussions of Burdett-FortiSandbox Combination cited by Dr. Black (Black Supp. Decl., ¶65)	Response
“Scanning service 304 cannot start processing a file until it arrives at the	There is no reason why the file would need to entirely arrive at the FortiSandbox device. The device does

FortiSandbox device, injecting delay into the system.”	not need to download an entire file, and it can begin scanning as soon as blocks of the file begin arriving.
“The additional network traffic from transmitting files from cloud infrastructure 302 to the FortiSandbox device consumes bandwidth and may result in network congestion.”	Again, the FortiSandbox device does not necessarily need to download an entire file. For example, if the first block of the file is sufficient to determine that the file contains malware, then the rest of the file no longer needs to be considered because it won't change the overall classification of the file.
“Transmitting files from cloud infrastructure 302 to the FortiSandbox devices results in unnecessary security risks, since there is always the possibility of a third party snooping traffic.”	If the network connection is encrypted, there is little risk of snooping. On the other hand, leaving the scan to be performed in the cloud can lead to security risks because an enterprise cannot be certain that the cloud scanner actually performs the expected functionality. A scanner under the

	operational control of the enterprise is much easier to supervise.
“There are additional managerial complexities in configuring permissions on both the FortiSandbox device and cloud infrastructure 302 to permit scanning service 304 on the FortiSandbox device to access files in cloud infrastructure 302.”	Any managerial complexities that result from the combination are likely desirable, or at least outweighed by other benefits, because, as discussed above, there are advantages to the enterprise having control over the scan.

39. Additionally, as described in the Petition and my declaration in support of the Petition, it would have been obvious to natively mount Burdett’s cloud file store to the FortiSandbox device at least because this combination would have enabled batch scans of entire directories of the cloud file store. (Pet., 30; Ex. 1002, ¶¶85-89.) These batch scans would have been more efficient than the file-by-file scanning taught by Burdett. (*See, e.g.*, Burdett, 28:41-45 (describing event notifications in the form of JSON files passed onto a queue, where “The scanning service may pull this JSON structure from the queue”); 29:31-34 (“Based on a notification of a file event...the scanning service may request a copy of the file to be examined.”)) For example, there would be a single (“push”) notification in the

combination that indicates a change to a directory. The FortiSandbox device could then scan the directory (“pull”) for any of the updates that occurred since the last notification – thus batching any file scans into one bulk operation. As I explained in my original declaration, the combination would be more efficient than Burdett’s process in which you have to wait for an individual push notification to arrive for each file before you can process it. The combination would reduce the number of notifications that are generated and would enable initiating individual file scans based on direct observation of changes in the file directory. (Ex. 1002, ¶¶52-56. 85-89).³ Dr. Black argued that information about “which files are new and need to be scanned...[can] easily be included in the notification, thereby allowing bulk scans.” (Black Supp. Decl., ¶64.) However, Dr. Black does not explain how the single-file notifications in Burdett would allow bulk scans.

2. Sharpe and Seese would have been obvious to combine

40. PO’s arguments against the Sharpe-Seese combination are similar to those against the Burdett-FortiSandbox combination, and I disagree with these arguments for similar reasons.

³Patent Owner also mischaracterized my deposition testimony on this point. See *infra*, ¶50.

41. First, PO argues that it would have been more efficient to “run Sharpe’s malware scans directly in the same cloud storage provider environment 4210.” (POR, 34.) However, there is no basis for this argument in Sharpe. Instead, Sharpe describes locating an anti-virus service at a remote, secure site: an anti-virus service “located at a secure site in close network proximity with the cloud storage system...[could] perform anti-virus scans for the distributed filesystem.” (Sharpe, 98:26-32; see also Pet., 53). Seese describes just such a remote, secure site that a POSITA would have found obvious to use with Sharpe. (Seese, [0018], [0064], [0067]; Pet., 63-64.) Additionally, Sharpe describes how a cloud controller “accesses the data blocks for the new file from cloud storage system 302...and the files are transferred to anti-virus services 4214.” (Sharpe, 97:48-51; emphasis added.) Since at least this portion of Sharpe already describes transferring files between different environments, it would not have been prohibitively inefficient to combine Sharpe and Seese by performing an anti-virus scan on a network security device.

C. In claims 8/19/27, the location of the second repository is merely an implementation decision

42. With respect to dependent claims 8, 19, and 27, PO makes a convoluted argument that it would not be obvious to be taking a file from a public cloud file store and, after determining that the file is clean, moving/copying that file to a

different storage location that is either part of the network security device or part of an external storage device that forms part of the enterprise network. (POR, 37-39.)

43. I disagree. A POSITA would have understood that files on a public file storage can be potentially malicious – we do not want these files inside of our network. But once these files have successfully passed examination, they are safe to store anywhere, including in the enterprise network. Any further details are merely an implementation decision.

44. Contrary to the PO's argument, modifying the references to locate the second repository on the network security device or on a device that forms part of an enterprise network indeed increases overall security, because only files that have been determined to be clean are allowed into the interior of the enterprise network. And counter to PO's argument that "there is no drawback to allowing the file to remain [in the public cloud storage] after it is confirmed cleaned," there may a number of operational arguments in favor of moving them to the interior of the network: inside the enterprise network, an enterprise would have much more control over the availability of the servers on which these files reside, and file access within an enterprise network in general might very well be considerably more efficient. I note that the PO has been making arguments about "efficiency" all over the place, except here.

D. The Sharpe-Seese-Kamalapuram-Dods combination rendered claims 3, 6, 11, 15, 17, 23, and 25 obvious

45. PO's only rebuttal to the Sharpe-Seese-Kamalapuram-Dods ground ignored Petitioner's explanation motivating the combination of references, and thus the Board should maintain its finding that claims 3, 6, 11, 15, 17, 23, and 25 are obvious. (*See* POR, 40; ID, 36-37.) As explained in the Petition and affirmed by the Board, Dods supplements the Sharpe-Seese-Kamalapuram combination to improve the combination's malware detection capabilities. (Pet., 80; ID, 36-37.) For example, Dods describes various malware detection techniques – including those claimed in claims 3, 6, 11, 15, 17, 23, and 25 – that are not explicitly discussed in Sharpe-Seese-Kamalapuram, but that were nevertheless well-known to those skilled in the art and which would have been readily used by a POSITA to perform the malware detection described in the Sharpe-Seese-Kamalapuram combination.

E. PO repeatedly misquoted or mischaracterized my testimony

46. PO's rebuttals are further undermined because their reply heavily leverages misquotations and mischaracterizations of my deposition testimony.

47. First, PO repeatedly asserts that, in response to their question “Does Burdett disclose the claimed copying?” I answered, “Burdett does not.” (POR, 1, 16, 20; quoting from Ex. 2004, 44:4-7.) However, review of the transcript reveals that this quotation misleadingly abbreviates my response in the middle of a sentence while failing to report the remainder of my response. My response continued:

“Burdett does not – first of all, what is the claimed copying? That – the patent uses these terms itself interchangeably, and Burdett teaches a mechanism that makes the clean file accessible only after the determination has been made that it’s clean.”

(Ex. 2005, 44:7-11.)

48. Next, PO suggests that I “[took] the position that *Burdett teaches moving not copying.*” (POR, 16, emphasis in original; citing Ex. 2005, 44:13-24 (“Burdett teaches enough that a person of ordinary skill would have found it obvious that one alternative to control access to a contaminated file was to *move* the file to a different location.”)) In this case, PO faithfully reproduced my words, but mischaracterized them as taking the position that Burdett teaches “moving” but not “copying.” I took no such position. Even suggesting that it would be possible to conclude that Burdett does not disclose “copying” from my statement that Burdett teaches “moving” contradicts PO’s position that “moving” can include “copying.” (See, e.g., POR, 17 (“Thus, while ‘moving’ can be implemented using ‘copying,’ ‘moving’ can also be implemented without any copying.”))

49. Similarly, PO seems to suggest that I *refuted* the Board’s finding that the ’856 patent uses “moving” and “copying” interchangeably when I replied, “I never said that” to the question “Is it your position that the claimed language ‘copying’ is synonymous with ‘moving a file?’” (POR, 18; citing Ex. 2005, 45:2-

4.) In fact, I expressly stated – immediately after PO ends their quotation – that “the patent specification uses these terms somewhat interchangeably.” (Ex. 2005, 45:4-5.)

50. With respect to obviousness of the Burdett-FortiSandbox combination, PO claims that I “[did] not allege [the bulk scan] leads to any advantage over the cloud infrastructure’s existing ‘security manager’ performing that scan and providing the pertinent information to the ‘network security device’ via the existing API.” (POR, 30; citing to Ex. 2005, 58:3-16.) The cited portion of my deposition responds to the following question from PO’s counsel: “Is there a difference in scan throughput, depending on whether the scan is performed by a scanning service with a natively mounted cloud file storage, or by the cloud file store itself.” (Ex. 2005, 58:3-6.) It is highly unreasonable to conclude from my answer to this question – that I “cannot answer that generally” about scan throughput – that I did not allege there to be *any* advantage over Burdett’s security manager performing a scan. In fact, there are numerous advantages to this, as I explained above.

51. PO then also claims I “admi[tte]d on cross-examination that Kamalapuram is silent on whether Kamalapuram’s ‘moving’ entails any ‘copying.’” (POR, 32; citing Ex. 2005, 68:8-10 (“‘moving’ might mean updating a directory...entry” or “[i]t might mean copying it to another disk and updating a directory entry.”)) Once again, even though they faithfully reproduced my statement,

PO's characterization of it as an "admission" that Kamalapuram is silent about copying is ludicrous.

52. Finally, PO misleadingly states that "as Dr. Franz confirmed in his deposition, Sharpe's malware scan (even without the teachings of Seese) was able to 'access files in the cloud file store' and 'control[] access to the files.'" (POR, 35; citing Ex. 2005, 66:9-19.) My actual statement, in response to the question, "Sharpe's scanning process was able to access files even without the benefit of Seese's teachings; correct?" was: "Sharpe had a way of accessing files *but didn't have all the benefits of the combination.*" (Ex. 2005, 66:9-13; emphasis added.)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001; and further that such willful false statements may jeopardize the validity of the application or any patent issued thereon. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on 3/8/24 in Irvine, CA.

A handwritten signature in blue ink, appearing to read 'M. Franz', written over a horizontal line.

DR. MICHAEL FRANZ

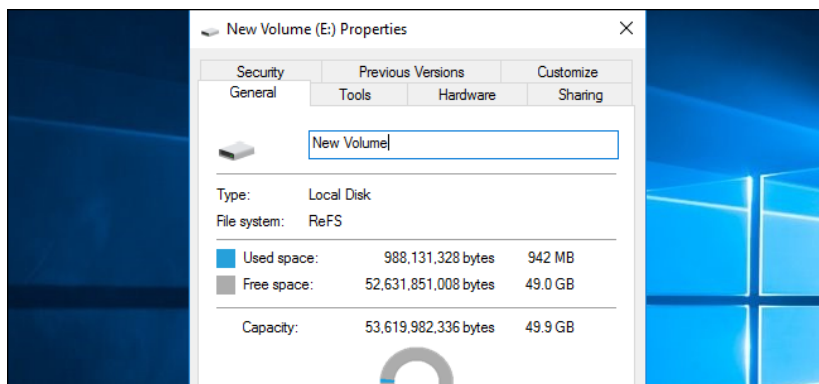
3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows?

The Wayback Machine - <https://web.archive.org/web/20170819005459/https://www.howtogeek.com/311840/what-is-refs-the-resilient-file-sy...>

What Is ReFS (the Resilient File System) on Windows?

by Chris Hoffman on June 26th, 2017

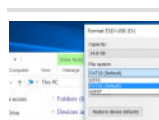


Microsoft's new ReFS file system was originally introduced on Windows Server 2012. It's included on Windows 10, where it can only be used as part of the drive-pooling [Storage Spaces feature](#). ReFS will be improved in Windows Server 2016, and will be part of Windows 10 Pro for Workstations.

But what is ReFS, and how does it compare to the currently-used NTFS?

What Is ReFS?

Short for "Resilient File System", ReFS is a new file system built using code from the current [NTFS file system](#). At the moment, ReFS is not just a replacement for NTFS. It has its own advantages and disadvantages. You can't just use ReFS instead of NTFS on your system drive.



RELATED ARTICLE

[What's the Difference Between FAT32, exFAT, and NTFS?](#)

As ReFS is Microsoft's newest file system, it's designed to address a few major issues with NTFS. ReFS is designed to be more resilient against data corruption, perform better for certain workloads, and scale better for very large file systems. We'll look at exactly what that means.

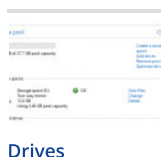
ReFS Protects Against Data Corruption

3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows

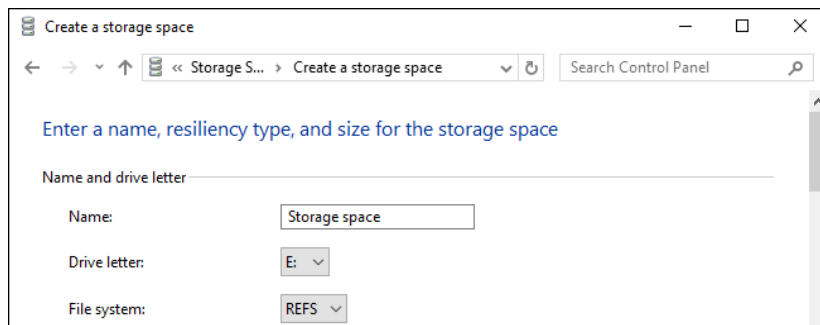
The “Resilient” part is highlighted in the name. ReFS uses checksums for metadata—and it can optionally use checksums for file data, too. Whenever it reads or writes a file, ReFS examines the checksum to ensure it's correct. This means the file system itself has a built-in way to detect data corruption on the fly.

ReFS is integrated with [the Storage Spaces feature](#). If you set up a mirrored Storage Space using ReFS, Windows can easily detect file system corruption and automatically repair problems by copying the alternate copy of the data on another drive. This feature is available on both Windows 10 and Windows 8.1.



RELATED ARTICLE

[How to Use Windows 10's Storage Spaces to Mirror and Combine Drives](#)



If ReFS detects corrupted data and doesn't have an alternate copy it can restore from, the file system can immediately remove the corrupted data from the drive. It doesn't require you reboot your system or take the drive offline, as NTFS does.

ReFS doesn't just check files for corruption when reading and writing them. An automated data integrity scanner regularly checks all files on the drive to identify and fix data corruption, too. It's an auto-correcting file system. You don't need to [use chkdsk](#) at all.

The new file system is also resistant to data corruption in other ways, too. For example, when you update a file's metadata—the file name, for example—the NTFS file system will directly modify the file's metadata. If your computer fails or the power goes out during this process, there may be data corruption. When you update a file's metadata, the ReFS file system will create a new copy of the metadata. ReFS points the file at the new metadata only after the new metadata is written. There's no risk of the file's metadata being corrupted. This is known as “copy-on-write”. Copy-on-write is also available on other modern file systems, like [ZFS and Btrfs on Linux](#) as well as Apple's new [APFS file system](#).

ReFS Drops Some Old NTFS Limitations

ReFS is more modern than NTFS, and supports much larger volumes and longer file names. In the long term, these are important improvements.

On an NTFS file system, file paths are limited to 255 characters. With ReFS, a file name can be up to 32768 characters long. Windows 10 now allows you to [disable this short character limit for NTFS file systems](#), but it's always disabled on ReFS volumes.

ReFS also discards the DOS-style 8.3 filenames. On an NTFS volume, you can still access C:\Program Files\ at C:\PROGRA~1\ for compatibility purposes with old software. These legacy file names are gone on ReFS.

3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows

NTFS has a theoretical maximum volume size of 16 exabytes, while ReFS has a theoretical maximum volume size of 262144 exabytes. That doesn't matter much right now, but it will one day.

ReFS Can Be Faster, Sometimes

ReFS isn't just designed to improve all performance over NTFS. Instead, Microsoft is focusing on a few important optimizations that make ReFS perform much better in certain cases.

For example, when used with Storage Spaces, ReFS supports "real-time tier optimization". You could have a drive pool with both drives optimized for performance and drives optimized for capacity. ReFS will always write to the drives in the performance tier, maximizing performance. In the background, ReFS will automatically move large chunks of data to the slower drives for long term storage.

On Windows Server 2016, Microsoft improved ReFS to offer better performance with certain virtual machine features. Microsoft's own [Hyper-V virtual machine software](#) takes advantage of these (and in theory, other virtual machine software could support them if they wanted to).

For example, ReFS supports [block cloning](#), which accelerates virtual machine cloning and checkpoint-merging operations. To create a cloned copy of a virtual machine, ReFS only needs to create a new copy of metadata on the drive and point it at an existing data on the drive. That's because, with ReFS, multiple files can point to the same underlying data on disk. When the virtual machine changes and new data is written to the drive, it's written to a different location and the original virtual machine data is left on the drive. This makes the cloning process much faster and requires much less disk throughput.

ReFS also offers a new "sparse VDL" feature that allows ReFS to quickly write zeros to a large file. This makes it much faster to create a new, empty, fixed-sized virtual hard disk (VHD) file. Whereas this might take 10 minutes with NTFS, it can take just a few seconds with ReFS.

ReFS Can't Replace NTFS (Yet)

All these features sound pretty good, but you can't just switch to ReFS from NTFS. Windows cannot boot from a ReFS file system, and requires NTFS.

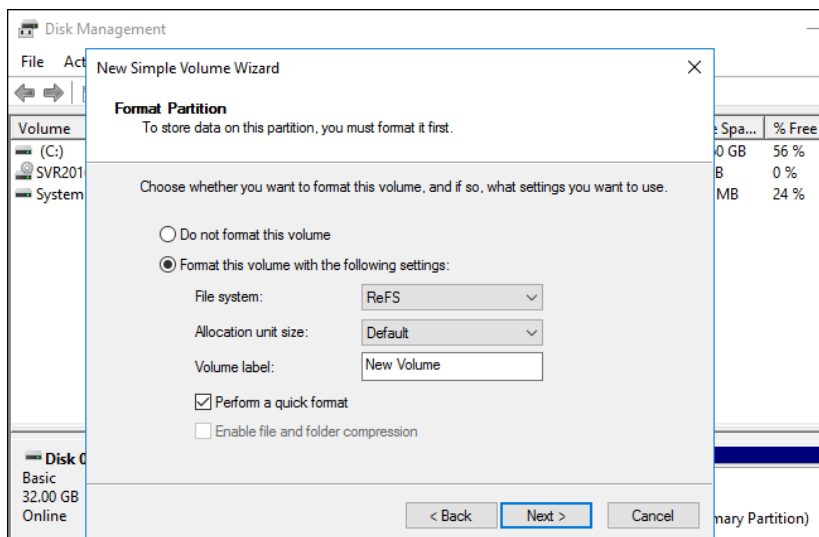
ReFS also omits other features NTFS includes, including [file system compression](#) and [encryption](#), hard links, extended attributes, data deduplication, and disk quotas. However, ReFS is compatible with a variety of features. For example, while you can't perform encryption of specific data at the file system level, ReFS is compatible with [full-disk BitLocker encryption](#).

Windows 10 does not allow you to format any old partition as ReFS, either. You can currently only use ReFS with Storage Spaces, where its reliability features help protect against data corruption. On Windows Server 2016, you can choose to format volumes with ReFS instead of NTFS. You might want to do this for a volume you plan on storing

3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows?

virtual machines on, for example. However, you still can't use ReFS on your boot volume. Windows can only boot from an NTFS drive.



It's unclear what the future holds for ReFS. Microsoft may one day improve it until it can fully take the place of NTFS on all versions of Windows. It's unclear when this may happen. But, for now, ReFS can only be used for specific tasks.

You'll find more details about ReFS and the specific features it supports on [Microsoft's website](#).

How to Use ReFS

On Windows 10, you can only use ReFS through the Storage Spaces feature. Be sure to format your Storage Spaces as ReFS and not NTFS when you see the file system option. You'll automatically benefit from the data integrity features in the ReFS file system if you choose it.

On Windows Server, you can choose to format some volumes as ReFS using the normal [disk management tools](#), which may be particularly useful if you're using virtual machines on those drives. But you can't format your boot drive as ReFS, and you'll lose access to some NTFS features.



3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows?

SHARE:



COMMENTS:

4

Chris Hoffman is a technology writer and all-around computer geek. He's as at home using the Linux terminal as he is digging into the Windows registry. Connect with him on [Google+](#).

DID YOU KNOW

The giant splashes of color that make poinsettias a popular holiday decoration are thanks to its colored bracts—the highly colorful leaves that surround the much smaller flower structure.

BEST OF HOW-TO GEEK



How to Install or Upgrade to Windows 10 on a Mac With Boot Camp



10 Quick Ways to Speed Up a Slow PC Running Windows 7, 8, or 10



How to Create ISO Files From Discs on Windows, Mac, and Linux



Yes, *Every* Freeware Download Site is Serving Crapware (Here's the Proof)



7 Ways To Free Up Disk Space on a MacBook



How to Use All of Windows 10's Backup and Recovery Tools



Wi-Fi Security: Should You Use WPA2-AES, WPA2-TKIP, or Both?



What's the Best Way to Back Up My Computer?



What Are Mesh Wi-Fi Systems, and How Do They Work?



GET THE HTG NEWSLETTER

Join 100,000 other subscribers

FOLLOW HOW-TO GEEK

3/8/24, 12:33 PM

What Is ReFS (the Resilient File System) on Windows?

160K+

50K+

90K+

MORE ARTICLES YOU MIGHT LIKE

ABOUT

About Us
Contact Us
Discussion Forum
How-To Geek Pro
Advertising
Privacy Policy

GET ARTICLES BY EMAIL

Enter your email address to get our daily newsletter.

GET ARTICLES BY EMAIL

FOLLOW US

TWITTER**GOOGLE+****FACEBOOK****RSS FEED**

Disclaimer: Most of the pages on the internet include affiliate links, including some on this site.

Copyright © 2006-2017 How-To Geek, LLC All Rights Reserved

[SIGN IN](#)

TECH —

macOS 10.12 Sierra: The Ars Technica review

Apple's desktop operating system once again plays second fiddle to iOS.

ANDREW CUNNINGHAM AND LEE HUTCHINSON - 9/20/2016, 10:00 AM

APFS: The next macOS (and iOS) file system

One of the highlight features of Sierra—if you're the kind of geek who loves internals and under-the-hood stuff—is [APFS, the new Apple File System](#). Though APFS is not yet feature-complete or ready for production use, Sierra ships with a developer preview version of the file system for the curious or brave user.

At some point between now and APFS' formal release (which will come sometime in 2017), we'll be taking a deeper dive into the file system, including both an evaluation of its features and some meaningful benchmarks. But to sate you until then—and to satisfy our own curiosity, since file systems are actually pretty exciting—we've been hammering on the beta for a few weeks. We've got a pretty good idea of what Apple intends to accomplish with APFS, and bugs and missing features notwithstanding, we like what we're seeing.

FURTHER READING

A ZFS developer's analysis of the good and bad in Apple's new APFS file system

Apple has restricted APFS in several ways in the Sierra beta, mostly to keep people from using it on their primary drives and suffering major data loss. There are six major restrictions right now on APFS volumes:

- You cannot use an APFS volume as a startup disk.
- You cannot backup an APFS volume with Time Machine (Sierra will allow you to select an APFS volume as a Time Machine destination to hold your backups, but you really, *really* shouldn't).
- You cannot encrypt an APFS volume with FileVault (though there are other native encryption options).
- You cannot use APFS with Fusion Drive.
- You cannot share an APFS volume over your network with AFP, which has technically been [deprecated since Mavericks](#). Sharing with SMB/CIFS or NFS is fine.
- APFS volumes are case-sensitive, no exceptions.

These restrictions apply so far only to the Developer Preview version of APFS included with Sierra for testing. Most of these restrictions (except the ones about AFP and FileVault) will be removed by the time APFS becomes a released product in 2017. Similarly, the file system's performance and its broad feature mix aren't necessarily representative of the finished product. For now, the APFS developer preview provides an illuminating peek into not just the way Apple's going to change its file system priorities in the future, but also into how it's going to try to shift some of the fundamentals of a user's experience with their computer.

But that's getting ahead of ourselves. Before we get into looking at APFS in Sierra, let's take a brief detour and make sure everybody's caught up on what's going on inside your computer's disk—and why a new file system is such a big deal.

What's on that disk?

Modern storage devices are a veritable turducken of nested layers of abstraction. There's no formalized equivalent of the [OSI model](#) for storage—the closest thing is probably [SNIA's network storage model](#), which still treats the underlying disk as kind of a black box—but in bastardized OSI terms, the file system sits at the storage equivalent of Layer 6, the presentation layer.

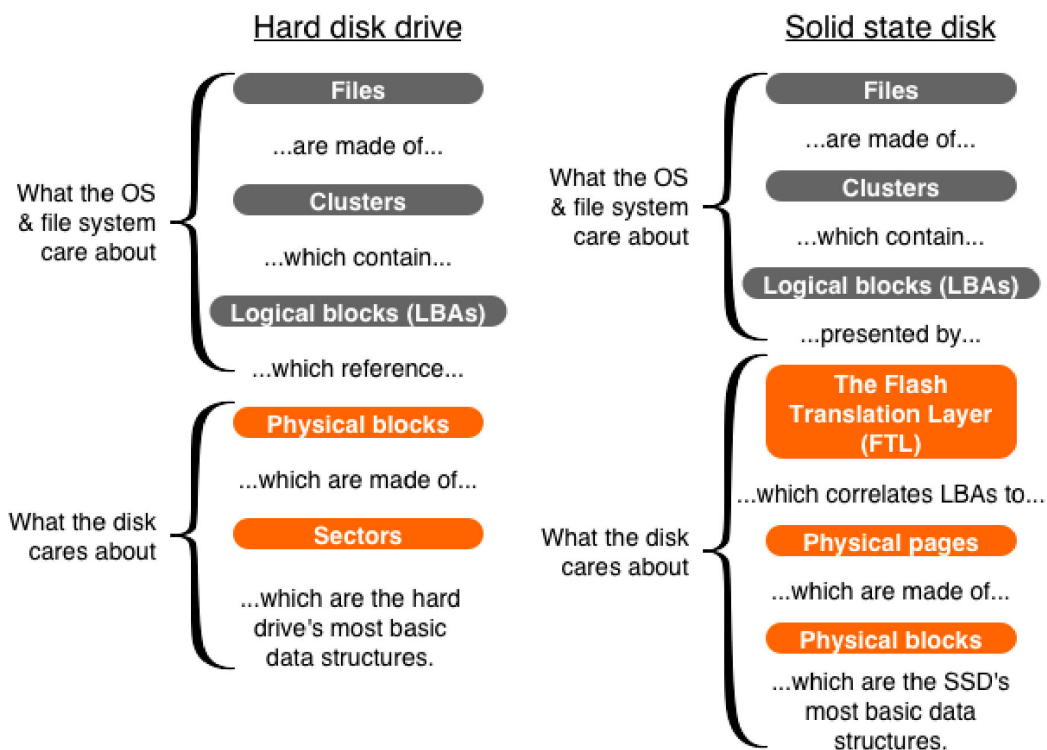
But there's tremendous logical separation between the top-level objects we as users interact with (primarily files and directories) and the actual physical structures they're composed of (NAND cells on a solid state disk, sectors on a spinning hard drive). The file system plays its part in abstracting away physical structure by correlating files to fixed-size chunks of space variously called "clusters" or "allocation blocks" or even (confusingly enough) just "blocks," depending on which file system you're using. Those chunks are then correlated with addresses on the disk called "LBAs," for Logical Block Addresses. LBAs are seen by the operating system as fixed, immutable locators for physical bits on the physical disk. However, as the name suggests, they're actually just another layer of abstraction.

(There's a complex historical reason behind why drives present LBAs to the operating system instead of "real" physical locations, but this review is long enough already without going down that particular rabbit hole. [Wikipedia is a good starting point](#) if you want to read about the reasons why old-school fixed block addressing was abandoned for logical block addressing.)

The file system doesn't care about anything lower than correlating clusters with LBAs—that's where the disk drive takes over, and what the drive does is of no concern to the file system. In the case of a spinning hard disk drive, the drive controls the mapping of LBAs to physical sectors. For solid state disks, there's even *more* abstraction, with something called the Flash Translation Layer constantly changing which physical pages of NAND cells are mapped to which LBAs.

FURTHER READING

Solid-state revolution: in-depth on how SSDs really work



A (very) simplified block chart of what goes where in the storage stack. Hard disk drives on the left, SSDs on the right. For simplicity's sake we're just calling file system allocation units "clusters" because that was short enough to easily fit in the diagram.

Years in the making

Apple has been making noises about changing the Mac's file system for more than a decade. For a long time, Sun's ZFS was the frontrunner to replace HFS+, right up to the point where **it wasn't**. Apple instead switched course to incrementally improving HFS+, most notably with **the significant addition of Core Storage in OS X Lion**.

FURTHER READING

ZFS: The other new Apple file system that almost was—until it wasn't

Core Storage (which is properly a logical volume manager) was a massive revamp to the underpinnings of how the Macintosh operating system dealt with its disks. It introduced a matryoshka doll-esque nested container model for combining volumes (either physical or logical) together into single big logical constructs. The most visible effect Core Storage has had on the Mac is that it enables Fusion Drive to exist. After **a splashy introduction**, Fusion Drive has been relegated to the background in Apple's presentations, but it's a "just works" kind of feature that binds two separate physical disks (an SSD and a hard disk) into one volume to ostensibly give you **the advantages of both**, and Fusion Drives remain a staple in mid-tier iMacs and Mac Minis.

But as noted by **John Siracusa a few years ago** in his Lion review, Apple's file system was aged and needing replacement even in 2007, and HFS+ with Core Storage

FURTHER READING

Achieving fusion—with a service training doc, Ars tears open Apple's

is still HFS+. It lacked a lot of buzzword features that were rapidly propagating out to other file systems, like snapshots, atomic writes, and checksumming data. It had a slow global lock design, limiting file system access to one process at a time. It could only write timestamps on files and directories in single second increments, which is a glacial age when it comes to file writes.

With ZFS out the window and Core Storage not addressing the totality of the HFS+ shortcomings, it seemed reasonable to assume that Apple was working on *something* file system related—and that brings us to APFS. Reportedly in the works for the past two years, **Apple says** APFS is intended for use as the file system on all its devices, "everything from an Apple Watch to a Mac Pro." Many of APFS' primary features are designed specifically around the strengths and limitations of NAND-based solid state storage, which in one form or another is the preferred option for every computing device the company sells these days.

So strap on your file system cap, folks, because we're wading in. Here's the list of what APFS brings to the table.

APFS features: Way the hell more files

APFS uses 64-bit inodes. An inode is a special file system block that contains information about files and directories. Think of an inode like a signpost on the disk—every file and directory has its own inode associated with it, and every object's inode holds metadata (location, timestamp, permissions, and other stuff) about the file or directory with which it's associated. File systems that use inodes all inherently have a maximum number of allowed inodes, and after you hit that number you can't create any more files or directories even if there's space left on the disk.

While HFS+ uses a 32-bit integer for its max inode count, APFS uses a 64-bit integer. This means the total number of file system objects goes up from about four billion in HFS+ to about nine quintillion in APFS.

What does this mean for you? This is a feature that dovetails with snapshots and clones. No one would reasonably expect to have nine quintillion files on a home computer—that's crazy. However, with APFS, inodes aren't just consumed by normal files and directories. When you begin throwing snapshots and clones into the mix—especially if Time Machine eventually shifts to using snapshots—the HFS+ upper bound of four billion inodes starts to become an approachable limit. Switching to 64-bit inodes pushes that boundary way, *way* out so you don't have to care.

Sparse files and lazy allocation

Funny names, simple concept: space needed by a file in APFS is claimed as it's actually needed, rather than all up front when a file is created. If an application needs to write a gigabyte-sized file that starts out mostly empty, for example, the empty space is reserved, but notionally represented by a single range of zeros instead of actually writing out all those zeros at once. Then, as the file is filled with real data, it "fills up" its allocation. (This is coupled with careful bookkeeping by the file system to ensure it doesn't over-allocate storage like an airline overbooking a flight).

What does this mean for you? This should mean means less time spent staring at an inscrutable beach ball if the operating system is tasked with large writes.

Process-level file system write locks

In order to prevent two applications from both writing to the same location on the disk at the same time—which would be bad!—file systems use the concept of *locking*. When an application or process goes to write to the disk, the file system *locks* that location so only the process that's going to write to that location is allowed to write to that location. When the write is done, the lock is lifted.

With HFS+, the lock is effectively system-wide—only one process at a time is allowed to write to the disk (more correctly, the file system's global metadata Catalog File can only be updated by a single process at a time, which in effect creates a system-wide IO lock). This isn't necessarily as problematic as it sounds, since these operations can be pretty darn fast, but sometimes there's a lot of reads and writes to deal with and they form a longer and longer queue, which makes the operating system sluggish or forces it to hang and show you a spinning beachball while the queue drains.

APFS institutes per-process locking so that instead of one system-wide hold, individual programs can ask for locks only on the areas to which they're writing. This means more concurrent IO and faster draining of the disk's queue.

What does this mean for you? As with lazy allocation, process-level locking is meant to translate into less time spent staring at the beach ball at random moments of high IO activity.

Write coalescing

It's always easier and faster to make one big write to a disk than it is to make a bunch of little writes. Every write operation has some amount of inherent overhead to it—the disk or SSD has to receive the instructions for the write, stop what it's doing, look up where the write needs to go, and then perform the write. With a spinning disk, this involves physically swinging the read/write head over the correct track and waiting for the correct sector or sectors to rotate underneath the head; on an SSD, the controller has to query the Flash Translation Layer and potentially even perform a lengthy block erase to free space to write (though not always).

Write coalescing is one of several enterprise features that APFS adopts to speed up disk access. Rather than performing a series of sequential writes as individual operations, they are *coalesced* into a single write and committed to the disk as one operation. This shaves down overhead, and the savings quickly add up when dealing with large numbers of write operations.

What does this mean for you? Yet again, this should be a beach ball-reducing feature.

Nanosecond timestamps

File and directory timestamps are often ignored by most people, but time metadata is vital to a file system from a perspective of *atomicity*—knowing *exactly* when modifications occur is critical to being able to effectively manage snapshots and clones, among other things. HFS+ can only record

timestamps down to the second, which is far too coarse to be useful. APFS brings timestamps that are accurate to the nanosecond—one billionth of a second.

What does this mean for you? On the surface, not much—it won't make any particular difference in your daily usage of your computer or watch or phone. But nanosecond timestamp granularity enables a lot of other advanced features on this list to function.

Crash Protection and Atomic Safe-Save

Writes to any file system take some non-zero amount of time—they're not instant, and power loss or any other kind of hiccup or interruption while a file system is writing to disk can be a Very Bad Thing because the hiccup can leave the file system in an invalid or inconsistent state. A half-written or corrupted file could have varying consequences depending on what file it is—from "I guess I just lost that document" to "I guess I'm reinstalling the operating system."

Modern file systems guard against this kind of thing in various ways, but the most common way is with *journaling*. The file system first documents the write it's about to make in a separate record (called a journal), then performs the write. If there's an interruption during the write, the file system can see that the journal doesn't match its file or directory, and it can (usually) recover from the problem by abandoning the write and pretending like it never happened. Journaling works well, but it creates a "double write" overhead because the file system has to first write what it's going to do to the journal, then actually perform the write for real.

APFS side-steps this by using a "copy-on-write" scheme when writing to the disk. Instead of first writing to a journal and then to the data part of the file system, APFS only makes one write to an empty area of the disk—even if the write is supposed to change an existing file. Once the write is committed and confirmed, APFS then updates the file or directory's inode to point to the changes. In this way, data is never overwritten—if the write succeeds, it succeeds and the updates are made part of the file. If it fails for any reason, the write is discarded without any changes to the original file.

Similarly, "Atomic Safe Save" provides the same kind of protection to whole-file operations ("atomic" here again refers to the computing concept of **atomicity**, not to some kind of nuclear super-save). While Crash Protection is a "bottom-up" technology concerned with ensuring individual logical block writes complete successfully, Atomic Safe-Save is a "top-down" tool with the goal of ensuring that all the operations involved with saving, renaming, copying, or moving files and directories all finish successfully, and without altering any original file data, before they're considered complete and the file is reported to you as saved, renamed, copied, or moved.

What does this mean for you? This should translate into far fewer opportunities for corrupted files—so fewer occasions to stare mournfully at the Disk Utility application while it tells you your stuff is gone.

Fast directory sizing

This one's easy—the method by which APFS checks the total size of a given set of directories (or even through the whole file system) has been changed so that instead of having to wade through the

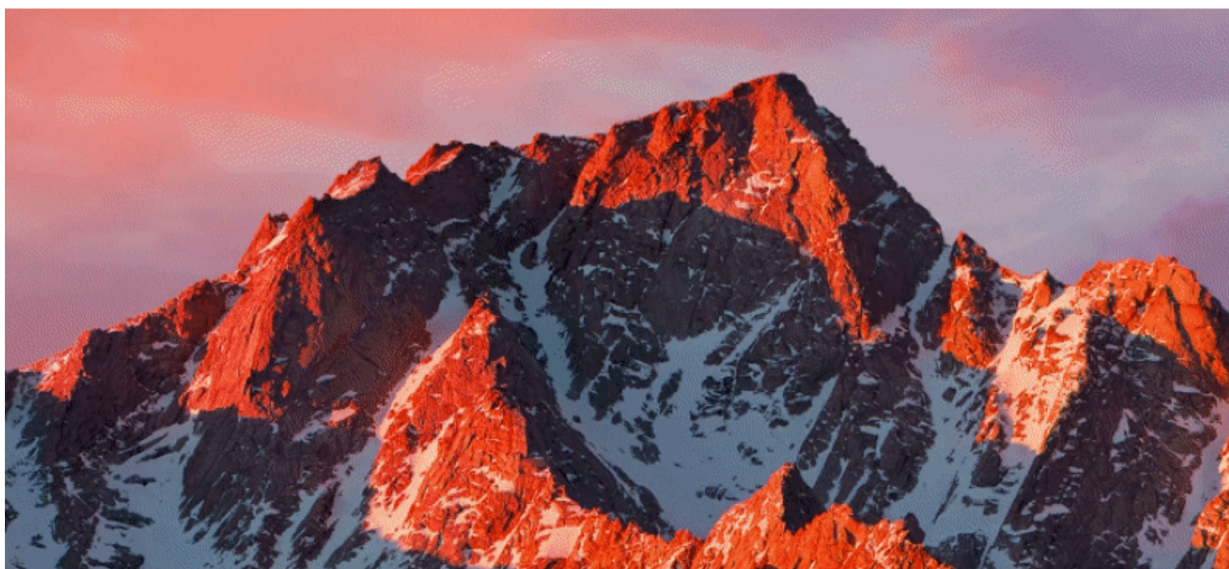
3/8/24, 8:02 AM

macOS 10.12 Sierra: The Ars Technica review | Ars Technica

whole collection of metadata for every file and directory object, file sizes are stored as separate metadata (which also side-steps a potential chicken-egg locking order violation problem you run into if you're storing file size metadata as part of file objects and you need to update a parent object with a child object's new size).

Consequently, reading the new file size metadata is a much faster operation than before—or at least it *should* be. We didn't have much luck with fast directory sizing in several rounds of testing in several different APFS containers. In fact, getting the size of a 53GB test dataset consisting of about 12,000 deeply nested files and directories took several seconds longer on APFS test volumes than it did on our HFS+ SSD (even when the APFS test volume was located on the same SSD as the HFS+ volume).

When asked, Apple informed Ars that any odd or inconsistent behavior observed in APFS—like fast directory sizing not being all that fast—should be chalked up to the prerelease nature of the file system. At least for now, that's an acceptable answer.



A short gif demonstrating how long it takes to perform a cmd-i "Get Info" task on a large folder on an HFS+ volume on SSD (left), the same folder on an APFS volume on SSD (center), and the same folder on a APFS volume on a removable USB 2.0 flash drive. The HFS+ folder reports its size immediately but takes a bit over nine seconds to display the total child object count, while the APFS folders show the size and object count at the same time after about five seconds. Interestingly, the slow USB 2.0 flash drive appears to finish first. This is likely because the HFS+ and APFS SSD volumes are actually hosted on the same SSD and the drive is performing both operations simultaneously.

What does this mean for you? Selecting a huge stack of directories and hitting option-cmd-I should result in a total size being displayed in a second or two, rather than many, many seconds—once APFS matures.

Native extended attributes

HFS+ introduced bolt-on support for extended attributes in **OS X 10.4 Tiger**. Extended attributes let the operating system store metadata other than standard file system metadata in a file or a directory.

Among other things, this provides a home for **resource forks** that accompany legacy Mac files. Extended attributes were also used for lots of other things, like determining whether or not a file sits under the macOS **System Integrity Protection umbrella**.

APFS provides native extended attribute support, so any files with extended attributes on HFS+ will keep them on APFS.

What does this mean for you? Again, not much in your day-to-day life. But if you happen to have old files that use resource forks (**and we know you folks are out there!**) or any applications that require extended attributes to function, this means they'll still work.

Space sharing

Space sharing is similar to the enterprise concept of "thin provisioning." It lets you allocate an APFS container on a disk, and then create multiple volumes inside that container, each of which can dynamically grow and shrink to take advantage of the whole container's amount of space. This way, you don't have to manually re-partition the volumes to shuffle free space from one to the other—they can resize themselves, up or down, as needed.

What does this mean for you? If you're the kind of person who uses multiple partitions on your disk, this means less time policing your partitions. You just throw them all inside a space-sharing APFS container and wash your hands of it.

Native encryption

APFS encryption is a big, *big* topic—so much so that we'll eventually be giving it its own article, once the file system is formally released. For now, the quick summary on APFS encryption is that rather than relying on Core Storage and FileVault for whole-disk encryption or third-party tools for per-file encryption, APFS supports three different kinds of native encryption, using either AES-XTS and AES-CBC ciphers depending on your hardware:

- No encryption
- Single-key file and metadata encryption
- Multi-key encryption

The last item, multi-key encryption, comes in three different levels of obfuscation:

- Metadata encryption
- Per-file encryption
- Per-extent encryption

Single-key encryption can effectively be whole-disk encryption, mimicking the functionality of FileVault today. Multi-key encryption, either of individual files or extents (an extent is a collection of APFS file system blocks), lets multiple users encrypt files (or even sections of files!) on the same volume. Alternately, it lets a single security-conscious user encrypt different objects with different keys. This is more-or-less **how storage encryption in iOS works today**—different "data classes" are

protected by different keys derived in different ways based on when and how information needs to be accessed.

There are as many potential use cases for native multi-key encryption as there are people who want to keep their things safe from prying eyes, and while it is an axiom that nothing is safe from a sufficiently determined attacker, it's *also* an axiom that sufficiently inconveniencing an attacker can make them move on to a softer target. Remember: if you find yourself standing next to a hobbit in front of a hungry dragon, you don't have to outrun the dragon. You just have to outrun the hobbit.

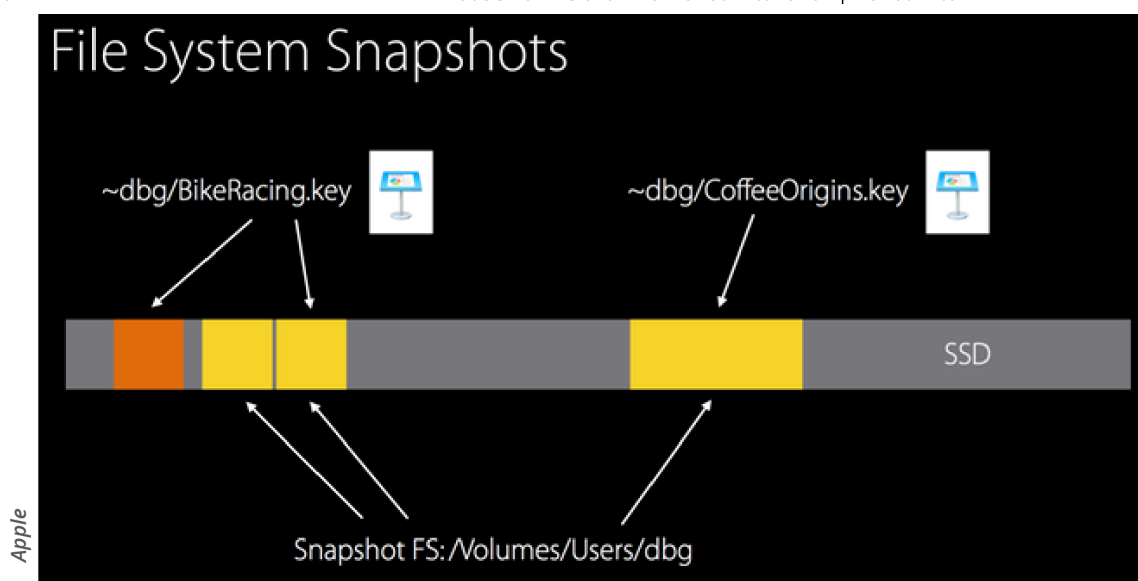
Native encryption is also a major reason why APFS doesn't do native **data deduplication**. However, snapshots and clones (see below) will help to at least partially fill the dedupe gap.

What does this mean for you? There are more choices for encryption, with single-key taking over the role FileVault has today. More importantly, APFS will bring a unified crypto architecture to macOS, iOS, tvOS, and watchOS.

Snapshots and clones

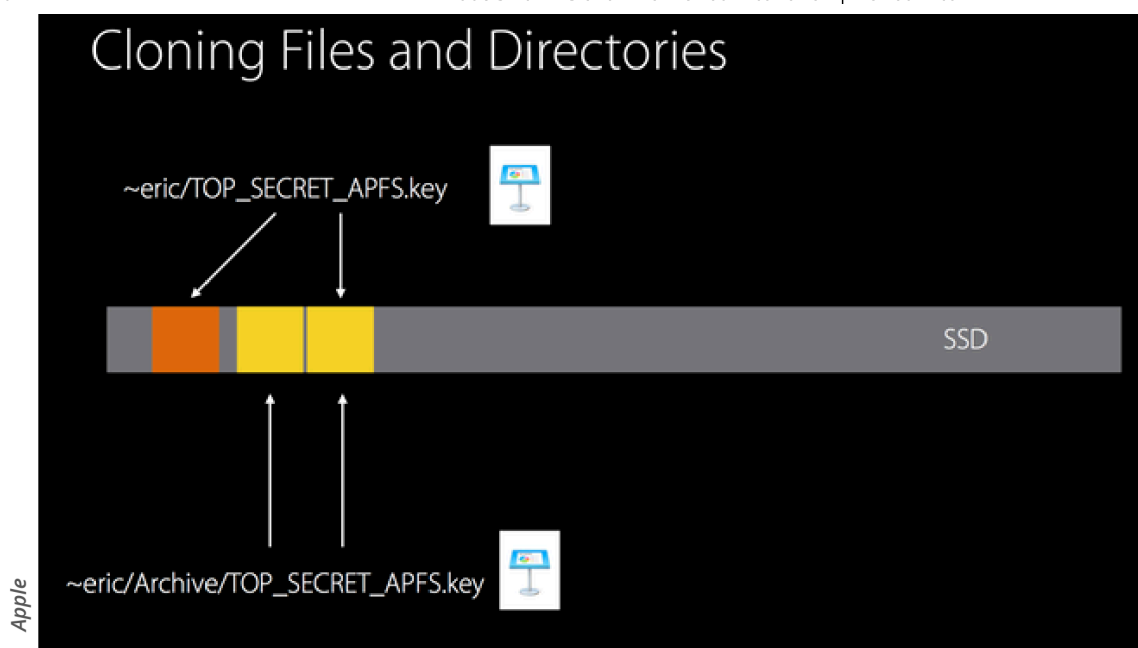
These are two of the more exciting APFS features. Both of them have to do with making instant copies of files, directories, or entire volumes, but the practical implementation of snapshots versus clones is different, as are the potential use cases.

Snapshots are read-only copies of an entire volume. They look and function like entirely separate volumes from the source, but they can be created instantly, as there is no waiting for anything to copy. As time passes and the source file system changes with adding files, deleting files, and the other things that happen during daily use, the snapshot consumes an amount of space equal to the total size of the changes. A brand new snapshot takes up no space, while year old snapshot on a busy file system might be very, *very* large. Deleting a snapshot deletes the record of changes between the time that snapshot was taken and now, freeing up space on the host volume. (If this sounds familiar, it's because this is essentially the same idea as snapshots on ZFS and other next-gen file systems, along with shadow copies on Windows).



A snapshot is a read-only copy of the entire file system. In this example image from Apple's WWDC 2016 presentation, both the original file system at top and the snapshot at bottom are made up of the same yellow blocks on the SSD. APFS keeps track of the differences between the read-write file system and the snapshot (like the changed orange block). Snapshots also track deletions, too—*any* change is held onto, so that at any point you can browse through the snapshot as if it were a complete version of the original file system frozen at one point in time. You can copy individual files out of the snapshot to the read-write file system, or even revert the read-write file system to the snapshot. Though Apple is mum on the subject, the implications for Time Machine are obvious.

Clones are a similar idea with a different focus—rather than being a read-only, frozen-in-time copy of a whole volume, a clone is an instantaneous copy of a file or a directory. Both the original file and its clone start out being made of the exact same file system blocks, but both are then treated as separate files and their content can diverge from each other. APFS knows which blocks are shared by the original and the clone, and which are added after the clone is created. The total amount of space consumed by a file and its clone is equal to the size of the original file plus the changes made post-clone to the original object and all of its clones.



Clones appear similar to snapshots at first, but clones are done with files and directories, not whole file systems. Further, cloned objects can be written to and edited. In this image, both the original (top) and clone (bottom) files start out being made of the same yellow blocks. The top file is edited and the orange block is added; the total amount of space consumed by both the original file and its clone is equal to the orange and yellow blocks.

There are some disadvantages to APFS' snap and clone implementation—chief among them being that snapshots and clones cannot be replicated or copied off-box, at least for now. One major use of snapshots on other file systems is to create a stable, non-active source or target for replication, and at least in its launch guise, there is no off-box functionality included in APFS' snapshots.

Both snapshots and clones use clever file system trickery to function. It all comes back to the way in which the underlying data structures are abstracted away from the files and directories we users interact with. This is a neat development, and one we're excited to start using—even if a lot of the use-cases for normal non-power-users have yet to be discovered.

What does this mean for you? From a day-to-day standpoint, probably not much, unless you're a developer who writes backup utilities or otherwise interfaces with the file system directly. The immediate impact is that when copying files around on the same APFS volume, the copy operation should take almost no time, because it's now automatically a clone operation. There are unanswered questions about how the introduction of clones will affect a user's attempts to clean up their hard disk—currently, there's no indication in the GUI or the command line to show whether a file is a clone or an original, or if it's an original with or without clones. Deleting a clone of a 10GB file won't free up 10GB of space—you'll have to delete the original *and* all clones to reclaim everything.

Apple's [APFS documentation](#) says "the operating system uses snapshots to make backups work more efficiently and offer a way to revert changes to a given point in time," so that's a non-subtle hint that there's a real possibility APFS snapshots will supplement (or even replace) Time Machine's existing house-of-cards pile of hard file and directory links. The synergy is obvious—hourly, daily, and weekly

3/8/24, 8:02 AM

macOS 10.12 Sierra: The Ars Technica review | Ars Technica

snapshots could easily provide the same functionality that Time Machine currently uses, but presumably with a much lower chance of backups becoming corrupted or unusable.

Page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) Next →

ANDREW CUNNINGHAM

Andrew has a B.A. in Classics from Kenyon College and has over five years of experience in IT. His work has appeared on Charge Shot!!! and AnandTech, and he records a weekly book podcast called [Overdue](#).

TWITTER [@AndrewWrites](#)

READER COMMENTS 311

SHARE THIS STORY

← PREVIOUS STORY

NEXT STORY →

Related Stories

Sponsored Stories

Powered by

Today on Ars

RSS FEEDS

VIEW MOBILE SITE

VISIT ARS TECHNICA UK

ABOUT US

CONTACT US

STAFF

ADVERTISE WITH US

REPRINTS

CNMN Collection

WIRED Media Group

Use of this Site constitutes acceptance of our User Agreement (effective 1/2/14) and Privacy Policy (effective 1/2/14), and Ars

3/8/24, 8:02 AM

macOS 10.12 Sierra: The Ars Technica review | Ars Technica

Technica Addendum (effective 5/17/2012). Your California Privacy Rights. The material on this site may not be reproduced, distributed, transmitted, cached or otherwise used, except with the prior written permission of Condé Nast.