The Old New Thing

If you let people read a file, then they can copy it

6 Dec 2006 7:00 AM | 67

Here's a question that floated past my view:

> How do I set the ACLs on a file so users can read it but can't copy it? I can't find a "Copy" access mask that I can deny. If I can't deny copying, I'd at least like to audit it, so I can tell who made a copy of the file.

There is no "Copy" access mask because copying is not a fundamental file operation. Copying a file is just reading it into memory and then writing it out. Once the bytes come off the disk, the file system has no control any more over what the user does with them.

Blog - Comment List MSDN TechNet

Comments

Adam
6 Dec 2006 10:11 AM
#

"Digital files cannot be made uncopyable, any more than water can be made not wet."

http://www.schneier.com/crypto-gram-0105.html#3

TKW
6 Dec 2006 10:17 AM
#

"Once the bytes come off the disk, the file system has no control any more over what the user does with them."

Surely you could say almost the same thing about executing a program, yet the "Traverse Folder / Execute File" ACE exists.

> [*That's right. If you you wrote your own program loader, you can write one that ignores the Execute File permission. -Raymond*]

Dave
6 Dec 2006 10:25 AM
#

"Once the bytes come off the disk, the file system has no control any more over what the user does with them."

Yes, but clearly we should make international laws that prohibit copying and then set a bit on the file that sends a message to the local government or its authorized deputies (MPAA, RIAA) any time the file is read. It just takes a slight enhancement to the current DRM.

> [*Sigh. I was hoping against hope that this wouldn't devolve into a flamewar about DRM. Took less than an hour. Okay, I'm letting everybody know: Any further comments to this post on the subject of DRM will be deleted. This is not a political blog. -Raymond*]

DavidE
6 Dec 2006 10:32 AM
#

There may be solutions to the actual problem, such as rendering the image to an image and allowing the user to read that image. At least that would prevent "casual copying", which is often an acceptable solution.

The real problem in this case is that the question assumes too much. The person attempting to solve the problem has made the leap to "ACLs are the solution". This is a common mistake made by many people. I've often received bug reports where the person writing the report tried to give a solution rather than clearly outlining the problem so that the people working on the product could decide on a solution that worked within the product design.

Adam
6 Dec 2006 10:34 AM
#

"If you you wrote your own program loader, you can write one that ignores the Execute File permission."

You don't even need to do that; again you can just copy the bytes of the file to a new file, mark the new file - your private copy - as executable, and presto!

But that's what execute-but-not-read (--x) permissions are for. Even your own program loader won't be able to do execute the file then as it can't read the file.

DavidE
6 Dec 2006 10:39 AM
#

The DRM rants make the assumption that the person is trying to prevent the copying for the purposes of IP protection. This may not be the case. There are situations where you just don't want people making copies of stuff because of workflow design. There are some users who need protection from themselves.

Of course, then there's the opposite situation. I recently downloaded a PDF of an application form that wasn't a PDF form. I had to print it out, fill it out, and then wait 5 minutes while the person at the office typed all of my information into their computer.

Ah, technology...

Frank Wilhoit
6 Dec 2006 10:46 AM
#

"Dave" was being sarcastic but what he says is what is going to happen, no matter what ghastly technological backflips will be needed to make it possible.

It is interesting, though, as well as a deep obstacle to generic analysis of security requirements, that different operating systems can have incommensurable notions of what the most primitive rights on a resource should be. One old system that I worked with (DEC TOPS-10) had separate rights for writing a file and deleting it. You could have write access to a file but not be able to delete it. The delete right was actually called rename because file deletion was implemented as a special case of renaming. It was done by changing the file name in the directory on disk to all NULs, which in turn triggered a background thread to reclaim the storage. I am sure that someone has wished over the years that some newer system would allow files to be writable but not deletable, or other unexpected permutations of "basic rights", which turn out to be not so basic after all but semantically richer than we thought.

andy
6 Dec 2006 11:28 AM
#

I guess the marketing/sales answer to that question is a link to "Windows Rights Management Services" (http://www.microsoft.com/windowsserver2003/technologies/rightsmgmt/default.mspx) :)

Adam
6 Dec 2006 11:33 AM
#

Frank - that's pretty much how Unix-type systems work, including Linux.

Writing to a file involves modifying the actual file. Removing a file involves modifying the directory that "contains" the file. So you need write permissions on a file to overwrite it or truncate it, but write permissions on the directory to delete it.

(It's a bit more complicated than that as files aren't really "contained" in directories, they just have links from zero or more directories and maintain a link count. When the link count reaches zero, the OS reclaims the disk space used by the file. Note links from _zero_ or more directories, as opening a file also increases its link count, so a file can exist in zero directories if it is open somewhere. Google around for more info...)

David Hunter
6 Dec 2006 11:41 AM

#

As long as you are performing a public service, you could try explaining this to the habitues on Second Life that are grumpy because "Copybot" is replicating their digital creations.

> [*They use NTFS ACLs in Second Life? From what you describe, it appears that their complaint is ethical not technological. -Raymond*

Bart
6 Dec 2006 12:07 PM
#

I'm allways affraid that a client gets angry in cases like this, complaining of going "technical".

That we should just keep it understandable and "do it".

Mike Dimmick
6 Dec 2006 12:12 PM
#

Actually I could really do with permissions so that users can write new files into a folder, but not edit or delete existing ones. This enables us to keep an archive of projects and versioned documents, removing the possibility that someone could accidentally or maliciously delete or move those files.

Our workaround at present is to have a single user which has permissions to write to those folders, and no other users. You then have to connect to the share as that user - which basically means logging on as that user, which I normally do by connecting remotely to a different box - in order to write to the archive. This has the obvious problem that the account's password has to be known by everyone who should have permission to write. I'm thinking of writing a program to automatically change that account's password on the same schedule as any other user has to.

Maybe I should be looking at SharePoint.

James Risto
6 Dec 2006 12:31 PM
#

At the risk of being off topic ... one thing I don't understand is that using NTFS-based permissions, a user gets different ACL's based on whether they move or copy a file. Move keeps the permissions. Copy inherits the destination folder's permissions. Confusing for normal users.

> [*If "copy" preserved permissions you could use it to circumvent quotas. -Raymond*]

Larry Osterman [MSFT]
6 Dec 2006 12:34 PM
#

Mike, you can do that - grant them create file access to the directory. The handle granted to the creator of an object has full control over the object, so they can open the file, write data to it and then close the file and they won't be able to modify it after it was closed.

Larry Osterman [MSFT]
6 Dec 2006 12:36 PM
#

James, that's because copy is "open source file+create destination file+read/write data from source to destination+close source/destination".

So copy's semantics are to create a new file, but rename's semantics are to move the file (including it's ACLs). Rename's semantics could be "Copy+Delete", but that's not the way it was defined.

Jade Philosopher
6 Dec 2006 12:59 PM
#

Yup. The issue here is a leaky (and, to some users, invisible) abstraction. To paraphrase Hayakawa, the name of the the permission is not the permission.

The names of the permissions seem to correspond to common user actions - reading a file, writing a file, deleting a file. Copying a file should "logically" be in that list. The permissions would more accurately be described as "Put these bits* onto the wires", "Alter these bits", "Alter those bits over there that point at these bits", respectively. Copying would then be "Put these bits onto the wires, read your mind, and alter some bits somewhere else in the world."

*Perhaps instead of "bits" I should be saying magnetic domains? On second thought, no, unless the file system is implemented on the read/write head of the hard drive.

kokorozashi
6 Dec 2006 1:00 PM
#

It seems the original question does describe a legitimate desire. The questioner is just looking to the wrong part of the system to provide it. She might be happy if there were a way to tell Explorer not to copy a file. Anyone with half a lick of expertise would be able to circumvent that, but if the goal is to prevent the clueless from shooting themselves in the foot, as opposed to DRM, this might help.

Reinder
6 Dec 2006 1:16 PM
#

A managed environment such as a JVM or .Net could have a separate kind of "unwritable memory", similar to 'tainting' in Perl, and restrict use of data in that memory to 'safe' calls such as drawing text to the screen.

The OS then could have a "no copying" flag on files that limits file reading to instances of that VM using 'tainted' calls. Writing a useful implementation of this that does not have security holes may be a bit of a problem, though.

microbe
6 Dec 2006 1:31 PM
#

: Actually I could really do with permissions so that users can write new files into a folder, but not edit or delete existing ones.

This can be done on standard Unix/Linux systems, more or less, with the "sticky" bit on the directory:

1. Any user can create new files

2. Any user can edit/delete his own files

3. Non-root users cannot edit/delete files owned by others

Archangel
6 Dec 2006 2:30 PM
#

I can sort of vaguely see where someone might think they want to implement read-but-not-copy, but it's obviously got so many flaws it's never going to work. 99% of users would just open the file, then find it's read only and get a "Save As" prompt, and save it elsewhere.

Of course, the genius of the people that come up with these things is that they don't see things like that...

Adam:

"Digital files cannot be made uncopyable, any more than water can be made not wet."

How about www.buydehydratedwater.com? ;-)

Darrell Wright
6 Dec 2006 2:50 PM
#

Novell had a flag like this. You could read a file but not copy it via the standard copy

command.

Anony Moose
6 Dec 2006 3:03 PM
#

Banning the "standard" copy program is pointless if someone can use other applications to read it into RAM then write the bytes out to another file.

For example, in UNIX, "cat filename > newfilename" would be a copy that doesn't use the "normal" copy utility.

I'm sure that you could do something similar with Novell. Security though hoping the users are completely ignorant? Heh, that's cute.

James
6 Dec 2006 3:38 PM
#

Dafter still, a certain server product offered "execute only" permissions; of course, the server has no way of knowing whether a client is requesting the file's contents in order to "execute" the contents or to "read", other than taking the client's word for it. (Even without actually modifying the client, KB157279 explains how to modify the "HKEY_LOCAL_MACHI"(sic) registry hive to stop this flag being honored.)

You get some fun discussions on this topic; one guy tried to prevent file copying (while allowing execution over the network) by removing Read permission, and was told this is a limitation of FAT32's file permissions... http://lists.ufl.edu/cgi-bin/wa?A2=ind0206&L=ccc&T=0&F=&S=&P=9519

Erzengel
6 Dec 2006 4:24 PM
#

Raymond: From a purely academic standpoint (as it doesn't really exist anyway), how does preserving permissions on copy circumvent quotas?

> [*Find a file owned by another user "X". Copy it. If permissions were preserved, the copy would be owned by X and therefore would be charged against X's quota. - Raymond*]

Nar
6 Dec 2006 4:40 PM
#

If you don't want people to copy data out of a secured area, you could set the ACLs so secure programs can't write out of that area.

BryanK
6 Dec 2006 5:11 PM
#

microbe: minor issue, but:

> 3. Non-root users cannot edit/delete files owned by others

Unless they own the directory. ;-)

Almost all world-writable-but-sticky directories are owned by root, so this doesn't usually come up, but it is something that's checked. The users that can remove files in a sticky directory are: (1) root, (2) the directory's owner, and (3) the file's owner.

Gabe
6 Dec 2006 5:40 PM
#

Raymond, you're confusing security descriptors with ACLs. The SD on a file contains the owner, the ACL, and other stuff. You could copy a file's data and its ACL without copying its owner or the other stuff in the SD.

In fact, it's not possible to copy the SD unless you have the Restore Privilege.

The other stuff in the SD is the group the object belongs to (for POSIX compatibility), the audit list and any mandatory integrity level (SACL).

Rod Widdowson
6 Dec 2006 6:19 PM
#

The flipside of this is that there is no IRP_MJ_CUT_AND_PASTE, much to the distress of many FS filter develeopers' managers

Philip Taylor
6 Dec 2006 8:17 PM
#

Somewhat related to what Reinder suggested, Compartmented Mode Workstations are one technology that does let users read but not copy data from files, to a certain extent - there's an example at http://www.ornl.gov/~jar/cmw4me.pdf of a windowing system where different title-bar colours represent different security levels, and data can't be copied down to a lower security level.

System V/MLS also had a windowing system with "mandatory-policy mediated and audited 'cut and paste'", as in http://www.acsac.org/waepssd/papers/01-flink.pdf . But that document tries to explain why the idea never took off despite being technically 'brilliant' (at least based on the designer's description of it) - users don't want that type

of security.

Sometimes they don't have a choice, like in military situations where confidentiality is critical and the system designers don't want users (or trojans) to ever send top secret satellite photos as attachments over public email. But otherwise it seems that, while some people may want to restrict others (e.g. by distributing uncopyable files) and agree with the general security principles, they won't accept the systems which are able to support that - they would have to suffer from the same restrictions themselves, and the inherent complexity and reduction in ease-of-use is such that a simpler, less secure (at a technical level), more usable system will fit their needs better. Or at least that's the impression I get.

Cooney
6 Dec 2006 8:20 PM
#

[Find a file owned by another user "X". Copy it. If permissions were preserved, the copy would be owned by X and therefore would be charged against X's quota. -Raymond]

I guess it's the unix in me, but permissions are generally distinct from ownership.

> [*True, ownership is different from permissions, but it's all part of the security descriptor. At any rate, it would be weird if you made a copy of a file and didn't have permission to modify or delete the copy! -Raymond*]

Igor
6 Dec 2006 8:29 PM
#

Actually there is a way to implement No Copy ACL.

I admit it might not be practical and it could surely be circumvented like any other protection but majority of users would not be able to do it.

First, it would be trivial to patch system CopyFile()/MoveFile() APIs so that they honor "No Copy" ACL.

Reading a file using ReadFile() API into memory should flag a memory pointer as a "No Copy". Attempt to call WriteFile() API using such pointer could then fail.

I am not sure about MMIO though.

> [*Watch my m4d h4cking skilz:*
>
> ```
> DWORD dw;
> BYTE b;
> while (ReadFile(h, &b, 1, &dw, NULL)) {
>   BYTE b2 = b; // s00prl33t! change the pointer!
>  WriteFile(h, &b2, 1, &dw, NULL);
> }
> ```
>
> *-Raymond*]

Adam
6 Dec 2006 8:29 PM
#

James - of course, for network filesystems, read/execute is pretty blurry. For local files though, the kernel knows whether a file is being opened for reading or for executing as opening for reading comes through "open()" or similar, while executing comes through "exec()" or similar.

(Sorry, I know this is a Win32 blog, but I don't know the equivalent "nt*" API calls off the top of my head.)

In the local case, the kernel *can* tell the difference and execute a file without allowing a user to read it. (assuming the filesystem can make the distinction, of course)

Igor
6 Dec 2006 8:59 PM
#

Raymond said:

"Watch my m4d h4cking skilz"

Hahahah, that was funny :)

Obviously that would need some serious hardware support too.

You could perhaps keep a hash for each "No Copy" file a process opens for reading and check a hash for each new file closed by the same process before commiting it to the permanent storage.

While we are at it, is there any way to get a directory change notification in Windows when a process closes a file? I couldn't find anything on the subject and you seem like the right hax0r to ask ;)

Igor
6 Dec 2006 9:03 PM
#

Oh and you could also forbid each process that opens a file marked as "No Copy" to use any WriteXxx() API.

steveg
6 Dec 2006 9:42 PM
#

Igor: and the clipboard, DDE, sockets, printf, and so on until all you're allowed to do is this:

__asm { NOP }

Save this one for the phillosphers, kids.

Cheong
6 Dec 2006 10:54 PM
#

[quote]

For local files though, the kernel knows whether a file is being opened for reading or for executing as opening for reading comes through "open()" or similar, while executing comes through "exec()" or similar.

[/quote]

So... What if a debugger steps in?

I think any debugger that "analyse" codes are "reading" that executable area. And the "minidump" program that drops memory image when program crashes are actually "copying" that memory to the disk. Invention of such feature would prevent those programs doing their job properly.

Any "secret" way to let those program pass will eventally let everyone bypass the protection.

ChrisR
7 Dec 2006 2:22 AM
#

>>Watch my m4d h4cking skilz:

Raymond for the win. Thanks Raymond, this was quite an interesting topic, and I had to comment that I cracked up when I read that.

Phylyp
7 Dec 2006 3:10 AM
#

>>Watch my m4d h4cking skilz:

Awesome!

James
7 Dec 2006 3:58 AM
#

(Sorry, I know this is a Win32 blog, but I don't know the equivalent "nt*" API calls off the top of my head.)

That's OK - the Native API (the Nt*/Zw* calls you mention) aren't Win32 anyway. FWIW, the open/exec equivalents (which use counted Unicode strings, which can contain embedded nulls and be case sensitive) are NtCreateFile and NtCreateProcess respectively; the Win32 functions - which, on NT derivatives, act as wrappers around the Nt* calls - are OpenFile and CreateProcess, with an A or W suffix for string type.

On the subject of restricted environments where you "can't" copy: apparently Cisco use something like that to keep the IOS source under lock and key.

Robert Moir
7 Dec 2006 4:45 AM
#

Yeah someone asked me how to do this a while ago...

My reply:

http://www.robertmoir.co.uk/images/Paranoid.jpg

Adam
7 Dec 2006 7:30 AM
#

Cheong > Most of those kind of problems have been solved for suid programs - just don't let the user do it.

A debugger would not be able to read the code pages (or probably any memory) of the executable. The minidump program, which I assume runs as a local administrator (or the part of the kernel that writes core files in the case of Unix) would make the dump file have the same owner as the original executable, without global read access.

So you can't debug every process on your system. That's the case anyway.

"Any "secret" way to let those program pass will eventally let everyone bypass the protection."

Are you saying that suid root programs cannot be secure? That's an ... unusual claim.

s00prl33t
7 Dec 2006 9:18 AM
#

>> You could perhaps keep a hash for each "No Copy" file a process opens for reading and check a hash for each new file closed by the same process before commiting it to the permanent storage.

DWORD dw;

BYTE b;

while (ReadFile(h, &b, 1, &dw, NULL)) {

BYTE b2 = ~b; // s00prl33t! change the pointer and not!

WriteFile(h, &b2, 1, &dw, NULL);

}

Just run it two times.

Cody
7 Dec 2006 10:03 AM
#

Raymond, I would ask you to put more "superleet" comments around, but then I realized that the rarity of the event is part of what made it so funny. King's to you, my man.

Phill W
7 Dec 2006 10:47 AM
#

>DWORD dw;

>

>BYTE b;

>

>while (ReadFile(h, &b, 1, &dw, NULL)) {

>

> BYTE b2 = ~b; // s00prl33t! change the >pointer and not!

>

>WriteFile(h, &b2, 1, &dw, NULL);

>

>}

>

> Just run it two times.

That is an interesting one to track and something that I've actually been thinking about how to solve for tracing execution in an emulator. I believe someone did actually succeed on an Atari 2600 emulator a while back.

However it would also be something that you wouldn't want to do in real time unless you had hardware support in the processor and ram. Something that I'm sure will eventually happen.

Phill W

Sean W.
7 Dec 2006 12:10 PM
#

> Watch my m4d h4cking skilz:

laugh Raymond, if you're ever in southeast Pennsylvania, look me up: I owe ya a free beer for that one :-)

Igor
7 Dec 2006 7:45 PM
#

You guys are not fair, you should thank me for inspiring Raymond to write such l33t Win32 code :)

>Just run it two times

Yes I know that is possible, as well as simple compression but it is not something Average Joe would be able to do easily.

I also said that you would most likely need hardware support. Consider this scenario:

If the CPU could mark the pages in memory as "No Copy", then any attempt to write out a byte read from such a page would result in say 0xC0000005 (Access Violation exception) UNLESS you write it into another "No Copy" page.

So if you want to decode WMA file you could open file with "No Copy" ACL, read it into an allocated memory region marked as "No Copy" (say VirtualAlloc() with PAGE_READWRITE | PAGE_NOCOPY), and then decode it into audio buffer provided by audio subsystem which has the same page attribute (PAGE_NOCOPY).

Now tell me how would you copy that WMA file?

Sure some hax0r could use another OS which doesn't support such protection in hardware but if the file has proprietary and strong encryption for which the source is not available then I really don't see any alternative to re-recording it to WAV on playback which of course reduces quality.

> [*How would you design a CPU that could detect that the 0x42 you're about to write out originally came from a "No Copy" page? And once you've designed that, how do you deal with context switches? -Raymond*]

Igor
7 Dec 2006 8:01 PM
#

"How would you design a CPU that could detect that the 0x42 you're about to write out originally came from a "No Copy" page?"

Because that 0x42 comes from the cache line which also has a "No Copy" bit and it is loaded in a shadow register which can write only to another "No Copy" cache line and

page?

Raymond says:

"And once you've designed that, how do you deal with context switches?"

Can you be more specific? How do you see them as a problem?

[*You load the 0x42 into the* al *register. An exception is raised. Now you have to save the registers into a* CONTEXT *structure on the stac... oh wait, the stack isn't No-Copy. And even if you get this fixed, it seems that the only thing you can do with No-Copy data is store it to another No-Copy page. You can't display it to the screen, you can't use it in a computation... what value is data you can't use? -Raymond*]

Igor
7 Dec 2006 8:26 PM
#

Raymond says:

"You load the 0x42 into the al register. An exception is raised. Now you have to save the registers into a CONTEXT structure on the stac... oh wait, the stack isn't No-Copy."

Why wouldn't it be, you don't write stacks out into a file or do you?

Raymond says:

"it seems that the only thing you can do with No-Copy data is store it to another No-Copy page."

Well not quite. You could perform any transformations you want on that data, compression/decompression, encryption/decryption, effects processing, whatever.

"You can't display it to the screen, you can't use it in a computation..."

Sure you can.

Hardware would allow you to write out "No Copy" data to a PCI/PCI-Express/USB/Firewire device which has video or audio class but _not_ to a storage class device.

Of course, someone determined enough could just reprogram an EEPROM on the PCI IDE/SATA controller to fake it as audio or video card and write their own custom driver for that but Average Joe would not be able to buy or download such a hack.

You just need to have the secure path for the data all the way.

[*My s00perl33t loop wrote the stack to a file. After I perform computations on No-Copy data, can I store it to non-No-Copy memory? If so, then my computation is going to be "x = x + 2 - 2". There, I've removed the No-Copy shadow bit from the value and can now write it to a file. If you say that the result of a No-Copy computation is also No-Copy and can only be sent directly to a video display, then this throws a wrench in nearly all decompression algorithms. (Besides, how does the CPU know whether a write to the address 0x1234 is going to a video device or a storage device? Are you also going to disable bus mastering?) Maybe with enough work you can get it to work, in which case I congratulate you on your PhD thesis. But this is not the place to develop your thesis. -Raymond*]

mirobin
7 Dec 2006 9:44 PM
#

What if any process reading a "no-copy" file was not allowed to open any sort of writable handle for the lifetime of the process? This would include preventing access to shared global member, transferring the data via com, etc.

Previously opened writable handles would be invalidated; alternatively, some sort of "nocopy compliant" bit could be set in the binary which causes all writeable handle creations to fail.

Not that I can think of many useful programs that can be written with such constraints (maybe a viewer app), but ...

Disclaimer: this comment is a thought excercise; I am not proposing that something like this should be implemented.

Cheong
8 Dec 2006 3:41 AM
#

[quote user=Adam]

Are you saying that suid root programs cannot be secure? That's an ... unusual claim.

[/quote]

No, I'm not saying that.

What I say is, the program that people will use to circumvent your "copy protection of executable part of memory" is most likely to be run in root/administrator or even localsystem if it needs to be, so the suid thing do not offer any protection to that. (unless one day microsoft announce that nobody can login as administrator and run anything as administrator, but that's very unlikely, if not impossible)

Cheong
8 Dec 2006 3:56 AM
#

mirobin: This may work, but what if the user of your application want to save "option" settings?

And you should also prevent other process from reading the viewer process in order to make this work, but than you're going to hit the problem about minidump and debugger I previously stated.

Cheong
8 Dec 2006 4:01 AM

#

Actually when I think deeper, any "no read" memory must not be marked as "executable" at all.

While you can write your viewer as "no read", any virus writer and also write their virus as "no read" to circumvent virus scanners protection. (The pattern matching is also a "read" action)

Adam
8 Dec 2006 4:19 AM
#

Cheong > Yeah, but if the attacker can run the debugger (or other arbitrary program) _as an Administrator_, you've already lost!

Cheong
8 Dec 2006 5:12 AM
#

[quote]

eah, but if the attacker can run the debugger (or other arbitrary program) _as an Administrator_, you've already lost!

[/quote]

Oops, it seems I've mixed your arguments with others.

It's true for your case.(For protecting things like business programs from copying out, because you have control on the machine the programs are running) But for others about using this to protect contents of viewers, the program used to circumvent the protection is likely to run as administrator so it's no use.

Adam
8 Dec 2006 8:45 AM
#

Ah. Yeah, mixing up peoples points on non-threaded discussions happens a bit to me too! :)

Philihp Busby
8 Dec 2006 9:10 AM
#

There's this certain directory on a unix mainframe at work that I find myself running into permissions issues. Certain things will only work when I'm the owner of this or that file... There's probably a reason for this.

For some reason, I'm not allowed to chown someone else's file and assume ownership, but I have read and delete access to it, and am allowed to create new files in the directory.

New files owned by me.

James
8 Dec 2006 9:57 AM
#

PB: Probably a 'sticky' (+t) directory?

On Unix, 'deleting a file' is actually an operation on the *directory* (you delete the *name*, rather than the file itself, then the file disappears once it has no name and nobody's using it any more). Editing files and changing their ownership or permissions, however, is an operation on the file itself: if it isn't yours, you might not be allowed to do something. So, you can have read access (because the file owner says so) and delete access (because the directory owner says so) but not anything else (it's not your file).

Windows is rather different in this respect. There's a quick search tool (for NTFS filesystems only) on my homepage (linked from my name) which takes advantage of this to find files more quickly than most approaches - this wouldn't work on Unix. At some point, I'll probably write a page explaining this...

Neil
8 Dec 2006 11:09 AM
#

Larry Osterman wrote:

>Mike, you can do that - grant them create file access to the directory. The handle granted to the creator of an object has full control over the object, so they can open the file, write data to it and then close the file and they won't be able to modify it after it was closed.

Apparently Novell discovered the hard way that some apps created a file to check that it had permission to create files, then reopened the file to put data in it, so they had to alter their create protection to include overwriting zero-length files.

Cheong wrote:

>So... What if a debugger steps in?

The debugger only sees the fixed up image in memory, so you would have to work out where e.g. the indirect jump tables are to translate them back into relocations.

Igor
8 Dec 2006 7:07 PM
#

Raymond said:

"After I perform computations on No-Copy data, can I store it to non-No-Copy memory?"

You can't. You can perform any transformation but you can only put the result into "No Copy" page again.

Raymond said:

"If you say that the result of a No-Copy computation is also No-Copy and can only be sent directly to a video display, then this throws a wrench in nearly all decompression algorithms."

How is that so? You can still alocate more "No Copy" output pages than you had them on input.

Raymond says:

"Besides, how does the CPU know whether a write to the address 0x1234 is going to a video device or a storage device?"

Well, CPU could simply attempt the write and PCI/USB/Firewire/whatever bus could then just reject the write cycle if it sees wrong device class on the other side.

Raymond says?

"Are you also going to disable bus mastering?"

Of course I wouldn't. Same principle as the above mentioned could apply to DMA controller as well.

"Maybe with enough work you can get it to work, in which case I congratulate you on your PhD thesis. But this is not the place to develop your thesis."

I am not doing any thesis on the subject because I am already past the university age (34) and personally I would never want to see anything like that implemented.

I just said that it _is_ possible and you ridiculed me because I didn't explain myself very well. I agree that this is not the place but I had to defend myself.

I apologize if I sounded like a mad scientist because when it comes to formal education I only have a high-school degree -- I am a self-taught ASM/C/C++/GPGPU programmer with a part-time job in medical imaging which involves code optimization and solid hardware knowledge.

mirobin
8 Dec 2006 10:44 PM
#

Cheong,

re: settings

I would expect your app would have to launch a separate process that would be responsible for managing settings. This is not terribly dissimilar to how Vista requires devs to split "admin" logic from "limited user" logic for a good LUA experience.

re: minidumps

Additional permissions are required to perform those operations; if you're able to perform a minidump on an arbitrary process, you're also able to take ownership of the file and remove the nocopy bit in the first place, right?

LongHairSteve
10 Dec 2006 12:44 AM
#

Many times I see Raymond making a comment that he 'can not believe that he had to say this'; which I always consider right on funny as heck.

Now, I can not believe I have to comment on this subject, because it should never have happened in the first place.

Raymond answered the why not question with three sentences. One would do, maybe more sentences would better clarify what we are talking about. But, absolutely his three are more than sufficient, and covers the essientials.

Folks, you have read and write controls over YOUR files only. If you agree to let another user on the same computer or another computer over the internet read your file, then your control does not extend into their computer files. This is the same as writing a book. Once published then anyone can hand copy and machine copy what was written for their own use. Intellectual property rights of words in a book are protected from

someone copying your work and reusing, especially, making profit from you work. Not protected against someone hand-coping or machine-copying your book for their own files of study of your book which they now own.

Now, some people claim their is a way to extend the control to my computer, and other claim you would need a PhD to make this happen. And to top it off some claimed that it would take a highly qualified computer expert like themselves (of course) to circumvent this control. People if you allow me to put up your file on my screen, dude, I can easily pick up my digital camera and take a picture of MY SCREEN!

Giving up read control of your file is sending a copy to my screen...and I be damned if you have any father rights to say what I do on what is displayed on my screen and my computer. You still control me from going into YOUR FILE and changing it. But if I want to refile my copy which you allowed me to read, then I will. Like a book, though, take me to court if I should republish your work as if it were my own.

enstiven
10 Dec 2006 7:23 PM
#

I really miss the option to specify an alternative owner for a user so that files created by the user is owned by someone else. I think you can do this in *ux.

Often you only want someone to be able to create\copy files to a location without them being able to access\change them afterwards. In Windows this is impossible:-(

LongHairSteve

11 Dec 2006 9:38 PM
#

Thanks, enstiven, for bringing up the unix aspect, as I forgot to cover that one. First, read and write is basic way computers operate, and not dependent on whether it is a Unix or Windows operating system.

If UNIX did prevent people from making copies after they allowed another computer to access their file; THEN NONE IN THE WORLD WOULD EVER WANT A UNIX COMPUTER! Because that would be a severe limit on the power of another computer, kind anti-thematic to the whole idea that computers can make working with documents better than pen and paper.

Realize, even though this concept of extended copy control does not exist, that there are ways to protect your Intellectual Property and make it available to others. Adobe creates and captures a market by giving away free document software readers, and making a profit off of people who then want to also have write and creation Adobe PDF software. Microsoft creates a market advantage by including a feature in their operating system that is clipboard program which makes copying files which your computer is allowed to read.

Get the idea -- with your files you DO have complete read/write control, and the option to give read control without giving up write control so that other people can make use of your shared file but can NOT change your file. It is basic, you have NO right to give up read control with a stipulation which limits the customer's computer power. Kind of like trying to tell someone that you can not fly to the moon because your propeller will not work in space!

Norman Diamond
12 Dec 2006 9:36 PM
#

The comparison with paper reminds me of an incident a few weeks ago. I downloaded a PDF document and could print all except the first page, but trying to print the first page resulted in an obscure error message. After a while I noticed that the first page had something resembling a letterhead. There was nothing secret about the first page, and the provider makes the same thing available on paper as they do on their web site, but just because the thing looked like a letterhead maybe that was why the printer or driver rejected it. This is a multifunction printer whose firmware probably rejects attempts to copy paper money too (though I didn't test it ^_^).

Ema
13 Dec 2006 11:14 AM
#

The bad thing (or ironic) I read here is discussion about this.

Should be obvious that if you can move any bit in RAM you can do what you want with it. Magic of computer science?

Norman Diamond
13 Dec 2006 8:21 PM
#

Just in time, this morning's newspaper reports that the author of Winny was found guilty. The judge said that the guy posted the program publicly on his web site.

The judge forgot to say[*] how Winny was copied from the programmer's web site to users' computers. Users were surely running Windows APIs, mostly from Internet Explorer but conceivably sometimes from Firefox or others. It's not completely obvious whether the server was running Windows APIs from Internet Information Server, Windows APIs from Apache, Linux APIs from Apache, etc. Anyway, a lot more than Winny's programmer participated in producing copying operations that provided the possibility of making illegal copies.

So why weren't all the programmers of the rest of those tools also found guilty? Oh yeah, because they weren't prosecuted. OK then, why weren't they prosecuted? The answer was posted a few days ago in a different thread.

[* or maybe the newspaper forgot to report the judge saying it, but I kind of don't think so.]

[*I don't see what this has to do with file permissions. If you want to editorialize on recent events, you can do it on your own blog. -Raymond*]

Norman Diamond
14 Dec 2006 2:56 AM
#

> I don't see what this has to do with file

> permissions.

True. I think more than half the comments on this entry, starting with the very first comment, discuss reading and copying but neglected the connection with file permissions. Sorry that I got caught up in it yet again.