

Re: Multiple users read/write to same directory in Linux via Samba

Source: <http://www.derkeiler.com/Newsgroups/comp.os.linux.security/2002-02/0758.html>

From: Lew Pitcher (Lew.Pitcher@td.com)

Date: 02/21/02

From: Lew.Pitcher@td.com (Lew Pitcher)

Date: Thu, 21 Feb 2002 16:23:15 GMT

On 21 Feb 2002 07:37:24 -0800, sam-nospam-@totallynerd.com (Sam Alexander) wrote:

>Can you explain what the umask is? Also, I'm familiar with the
>User/Group/Other and Read/Write/Execute permissions, but what's the
>other permissions you guys are referring to? Also, I've done `chmod 775`
>and so forth, but in `3775`, what's the 3 for? I've never done a `chmod`
>with 4 digits.

OK, let's start at the top...

There are three classes of users affected by a file's permission flags

- 1) the user who owns the file,
- 2) the users who are in the same group as the group that owns the file, and
- 3) everyone else

There are three permission flags for each of these classes of users:

- 1) permit or deny 'read' access to this file,
- 2) permit or deny 'write' access to this file, and
- 3) permit or deny 'execute' access to this file

In addition, there are two flags that augment the execute privilege:

- 1) permit or deny the program to execute as if it had been run by the owning user (this is the 'setuid' flag), and
- 2) permit or deny the program to execute as if it had been run by a user in the owning group (this is the 'setgid' flag)

There's another permission flag (the 'sticky' flag) which (although it is present, and is useful under certain circumstances) I won't get into.

Now, each of these permission flags is implemented as a single bit in a binary number, with the bit weights assigned as follows:

2^{11} = 'setuid' flag

2^{10} = 'setgid' flag

2^9 = 'sticky' flag

2^8 = read permission for the user

2^7 = write permission for the user

2^6 = execute permission for the user

2^5 = read permission for the group

2^4 = write permission for the group

2^3 = execute permission for the group

2^2 = read permission for others

2^1 = write permission for others

2^0 = execute permission for others

These bits can be expressed as a string of 4 octal numbers:

- the high order octal number is the setuid/setgid/sticky flag (assumed 0 if omitted)
- the second octal number is the user rwx permissions
- the third octal number is the group rwx permissions, and
- the fourth (low order) octal number is the 'other' rwx permissions.

So, permission value 0751 gives

- no setuid
- no setgid
- no sticky
- user read
- user write
- user execute
- group read
- no group write
- group execute
- no other read
- no other write
- other execute

permissions to the file.

With me so far?

Now, these permission bits are initially given to a file when it's created. The values are explicitly assigned by the program that creates the file, as part of the system call that's used to create (and open) the file first. However, as a user, I might not like the permissions that a program gives to it's created files; the compiler gives 0755 permissions to a.out files and I (being the paranoid person that I am) want a.out files to only have 0700 permissions.

The umask gives me some control over which permission bits actually get used when programs that I run create files. Actually, the umask is a list of permission bits that I *don't* want to be set. If I don't want anyone else to read/write/execute any of my programs (I want 0700 instead of 0755 on a.out files), my umask would flag the group read, write, and execute flags, and the 'other' read, write, and execute flags. In other words, my umask would carry a

comp.os.linux.security: Re: Multiple users read/write to same directory in Linux via Samba
value of 0077.

When the creat() system call is invoked by a program that I run, the file permission bits passed to the call by the creating program are ANDed with the complement of the umask (permission & ~umask), and the resulting value is used as the permission bits for the file. In my example, this means that...

```
umask(0077);  
creat("somefile",0755);  
results in "somefile" getting the permission bits (0755 & ~0077)  
0755 & ~0077 = 0755 & 7700  
= 0700  
= user read/write/execute, no group/other read/write/execute
```

umask hangs off of your userid (for the lifetime of your connection), so it affects all files from all programs you execute from that point on.

Simple, isn't it <vbg>

BTW, 'man 1 chmod', 'man 2 chmod', and 'man 2 umask' gives a lot of this information.

[rest snipped]

Lew Pitcher

IT Consultant, Development Services

Toronto Dominion Bank Financial Group

(Opinions expressed are my own, not my employers')

- **Next message:** [Tim Haynes: "Re: Server Compromised through SSH or...?"](#)
- **Previous message:** [Nico Kadel-Garcia: "Re: Multiple users read/write to same directory in Linux via Samba"](#)
- **In reply to:** [Sam Alexander: "Re: Multiple users read/write to same directory in Linux via Samba"](#)
- **Next in thread:** [Sven Vermeulen: "Re: Multiple users read/write to same directory in Linux via Samba"](#)
- **Messages sorted by:** [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)