

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

Part I

It recently occurred to me that in the eight years or so I've been writing this column, I've never covered file servers. I've covered secure file *transfer*, for example, via scp, rsync and vsftpd, which certainly is important. But, I've not covered file serving, specifically, allowing users to mount persistent "network volumes" that let them use networked server disk space as though it were a local disk. This has all sorts of productivity- and operations-related benefits: it's (usually) easy for end users to use, and it makes data easier to access from multiple systems and locations and easier to back up and archive.

As it happens, there's a rich toolkit available to Linux users for building, securing and using file servers, mainly in the form of Jeremy Allison and Andrew Tridgell's Samba suite of daemons and commands, plus various graphical tools that supplement them. For the next few columns, I'm going to show you how to build a secure Samba file server using both command-line and GUI tools.

Does that sound like a good Paranoid Penguin project? Good enough, I hope, to forgive me for ignoring file servers for so long. (So many things to secure, so little time!)

What We Want to Achieve

Obviously, no series of articles can cover everyone's file server needs or wants. So, before I begin, let's agree on some requirements of my choosing (hopefully, some or all of these coincide with yours). It seems reasonable to focus on the following: security, convenience and cross-platform compatibility. It goes without saying that in a security project, security is foremost among my preoccupations. With a file server, I'm going to pay particular attention to protecting the data itself: the integrity, availability and confidentiality of my files, while at rest on the server and in transit over the network. I also want to protect the server itself, both to protect the data and to prevent the server from being misused in other ways.

The trio of goals I listed above (confidentiality, integrity and availability) is part of classic information security dogma. In just about any information security scenario you can think of, C, I and A are important one way or another.

The goal of simply preventing a system from being used in unexpected or unwanted ways by unauthorized persons though, is what I like to call exclusivity. If I go to the trouble of building and maintaining a file server, even if the data itself is 100% boring and useless (please do not insert a joke about Paranoid Penguin column archiving here), I want such a server to be used *exclusively* by me and the users I specifically designate.

Even if it's some sort of public file server (for which, by the way, asynchronous file transfer technologies, such as FTP, HTTP and rsync are *much* more securable than Samba), I still want that server to be used *exclusively* for that purpose. I don't want it being used as someone's pirate IRC server, warez repository or proxy for attacking other systems.

I just mentioned that the type of file server I'm talking about (the kind to which you can "map drives") isn't suitable for public file serving. This is because the two dominant tools for this, Samba and NFS, historically have relied on RPC, a protocol that involves the dynamic assigning of TCP and UDP listening ports on a per-client, per-connection basis, which requires a large range of ports to be

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

opened through any firewalls that might be in the way. Alas, opening UDP and TCP ports 1025 through 65,534 in both directions through a firewall is an awful lot like not using a firewall at all, even if you limit source or destination IP addresses.

On the one hand, more current versions of NFS (versions 3 and 4) allow the server/dæmon to use a single TCP port for all connections by concurrent users. However, much of the world seems to be stuck on NFS v2. Worse still for our purposes here, there never has been good support for NFS outside the world of UNIX and UNIX-like platforms.

And, this brings us to our other two requirements: convenience and cross-platform compatibility. I want to be able to map network drives/volumes, because this is much, much more convenient than manually copying files to and from the file server every time one changes, even automatically via some script. I want a file share that allows me to “work” on files that “reside” in a central location; I don't want working copies of my files being maintained on umpteen different systems.

I've alluded to the fact that NFS allows you to map network volumes in a very similar way as with Samba. However, I have to acknowledge the ugly reality that I am sometimes required to operate Windows systems. My job requires it and so does my video-gaming habit. So, I need a file server that supports both Linux and Windows clients. (As it happens, the one we're going to build also will support FreeBSD, NetBSD, Solaris, Mac OS X and practically the entire rest of the *nix world!)

To summarize, we're going to build our file server with Samba, because it's convenient and it supports different client platforms. And, we're going to build it as securely as possible.

Samba over the Internet?

If Samba is so very convenient, you may wonder, why not use IPsec or some other VPN/encryption tool to secure its use on the Internet? This is actually possible. As it's been a while since I covered IPsec in this space, and in the intervening years, IPsec support has been added to the Linux kernel, I just may end this series with a quick tutorial on doing Samba over IPsec.

However, Samba is a very “chatty” protocol—it generates a lot of packets even if you're using it only for small files or shares. This causes problems not so much for your Internet link as for Samba performance: Samba can be very sensitive to dropped or delayed packets, which is more likely in your modestly sized Internet uplink than over your exponentially bigger Local Area Network (LAN) fabric.

So, trying to get Samba working over IPsec may or may not be worth your time, and it may or may not warrant my covering it in this series of columns. Have no fear, one way or another, you can expect me to provide a tutorial on using the Linux kernel's IPsec functionality in a future Paranoid Penguin column.

Samba Security Terms and Concepts

I've explained in gory detail why Samba, firewalls and the Internet don't go well together. So, how do you secure Samba for LAN use?

Samba security is a surprisingly complex topic, which is why this is a multipart article. You've got many choices to make if you want to use Samba securely. Is your Samba server also going to be an

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

NT domain controller, or will it participate in an existing domain or workgroup? Will you permit guest access, or will all users of every share need to be authenticated first? Or, will you allow both private and public shares?

Don't worry if the previous questions make little sense to you. That's why the Paranoid Penguin is here. For most of the rest of this article, I discuss these concepts in detail. Only then will you be ready to explore the mysteries of smb.conf and the NMB daemon.

First, let's get some definitions out of the way:

- SMB: the Server Message Block protocol, the heart of Samba. SMB is the set of messages that structure and use file and print shares.
- CIFS: short for the Common Internet File System, which in practical terms is synonymous with SMB.
- NetBIOS: the API used to pass SMB messages to lower-level network protocols, such as TCP/IP.
- NBT: the specification for using NetBIOS over TCP/IP.
- WINS: Microsoft's protocol for resolving NBT hostnames to IP addresses; it's the MS world's answer to DNS.
- Workgroup: a peer-to-peer group of related systems offering SMB shares. User accounts are decentralized—that is, maintained on all member systems rather than on a single central server.
- NT domain: a type of group consisting of computers, user accounts and other groups (but not other domains). It is more complex than a workgroup, but because all domain information is maintained on one or more domain controllers rather than being distributed across all domain members, domains scale much better than workgroups.
- Active Directory: Microsoft's next-generation domain system. Samba can serve as an Active Directory client via Kerberos, but you can't control an Active Directory tree with a Samba server as you presently can do with NT Domains. Active Directory server support will be introduced in Samba v4.
- User-mode security: when a Samba server's shares are authenticated by local workgroup user names and passwords.
- Share-mode security: when each share on a Samba server is authenticated with a share-specific password that isn't explicitly associated with a user name.
- Guest access: when a Samba server allows anonymous connections to a given share via a shared guest account with no password.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

Here's what you need to take away from that list of definitions.

First, the protocols. SMB, aka CIFS, is the protocol that defines the network filesystem—its structure and its use. NetBIOS provides an API through which SMB messages may be transmitted over networks, and which may be used by servers to “advertise” services and by clients to “browse” those services. NetBIOS can use any of a number of lower-level network protocols as its transport, but the most important of these is TCP/P; NetBIOS over TCP/IP is called NBT. WINS provides centralized name services (mappings of hostnames to IP addresses), where needed.

Next, server roles. A Samba server can authenticate its transactions either on a per-share basis, using share-specific passwords and inferred/implicit user names, or on a per-user basis, using either a dedicated local user database (in user-mode security) or some networked authentication scheme, such as LDAP, NIS, NT Domains or Active Directory. The server can be in a workgroup, in which case it needs to maintain its own database of all the workgroup's user information, or it can be in an NT Domain or an Active Directory, in which all user information is managed centrally.

When you want to share data with maximum convenience and minimum security, for example, read-only files containing nonsensitive data, you can put it on a share with guest access. Users connecting to such a share will not be prompted for any user name or password.

The bad news is that this is only a fraction of what you need to know in order to understand SMB/Samba services. The good news is, NT Domains and Active Directory are out of scope for this series of articles. We're going to focus on using workgroups for our secure Samba file server.

Workgroups don't scale well, because each server in a workgroup needs to maintain all user information for the entire workgroup, and you must somehow keep this information (passwords and so forth) consistent across all workgroup members (except where only guest access or share-mode access is permitted).

However, for the usage scenario I've described—creating a file share or two I can reach from anywhere in my house—I'm not going to have very many users or even more than one server, necessarily, and the simplicity of setting up a standalone/workgroup server trumps the complexity-laden power that comes with NT Domains. If your needs differ, hopefully this series of articles nonetheless will make it easier for you to figure out Samba's NT Domain support on your own, if that's what you really need instead.

So, to express our project in the terms I've just defined, in subsequent articles, I'm going to walk through the process of configuring a standalone (workgroup) Samba server operating with user-mode security, using a dedicated local user database. Our example server will host a combination of guest shares, read-only shares restricted by user and shares that can be read by only some users, but that can be written to (changed) by others.

First though, we have to make sure you've got the software you need in order to pull that off.

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

Getting Samba Software

On your Samba server, you're going to need your distribution's packages for Samba's libraries; the Samba daemons `smbd`, `nmbd` and `winbindd`; the Samba client commands `smbclient`, `smbmount` and so forth (which are useful even on servers for testing Samba configurations); and also the Web-based configuration tool SWAT (Figure 1). Naturally, nearly all these things are contained in packages whose names don't correspond neatly with the names of their component daemons, libraries and so forth, but I give some pointers on those shortly.



Figure 1
SWAT

First, a word about SWAT, which requires a modest security trade-off for Ubuntu users. Although normally in Ubuntu the user `root` can't log in directly, Samba requires this to be possible, so you need to set a root password on any Ubuntu box that runs SWAT.

Like so much else about Samba, this is not something I recommend doing on any Internet-facing Ubuntu box. However, SWAT is such a useful and educational tool, I feel pretty confident in stating

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

that in non-Internet-facing environments, the mistakes SWAT will help you avoid probably constitute a bigger threat to system security than SWAT does.

As I mentioned, Samba packages are included in all major Linux distributions. In Debian and its derivatives, such as Ubuntu, you'll want to install the following deb packages: samba, samba-common, samba-doc, smbclient and swat (plus whatever packages you need to satisfy dependencies in any of these).

In SUSE, you'll want to install samba, samba-client, samba-winbind and samba-doc. (SWAT is included with one of these, probably samba.)

In Red Hat Enterprise Linux and its derivatives, you need samba, samba-client, samba-common and samba-swat.

Installing these binary packages should involve installation scripts that put startup scripts, symbolic links and so forth in the correct places for everything to work (at least, after you configure Samba to serve something). Using SWAT is the best way to get up and running quickly—not because it does very much work *for* you, but because its excellent help system makes it super-convenient to summon the pertinent parts of Samba's various man pages.

There are two SWAT quirks I should mention. First, SWAT must be run by an Internet super-server, such as the old Berkeley inetd or the newer xinetd. Ubuntu configures inetd automatically when you install the swat package, but if your distribution of choice does not, you need a line like this in `/etc/inetd.conf`:

```
swat  stream  tcp    nowait.400  root    /usr/sbin/tcpd  /usr/sbin/swat
```

Second, to get SWAT's help links to work under SUSE 11.0, you may need to create the following symbolic links while logged in to a terminal window as root:

```
In -s /usr/share/doc/packages/samba/html/docs/manpages /usr/share/samba/swat/help
In -s /usr/share/doc/packages/samba/html/docs/using_samba /usr/share/samba/swat/help
In -s /usr/share/doc/packages/samba/html/docs/index.html /usr/share/samba/swat/help
In -s /usr/share/doc/packages/samba/html/docs/manpages.html /usr/share/samba/swat/help
```

Conclusion

And with that, we're ready to start configuring our Samba server! Or we would be, if we weren't out of time and space for this month. The links in the Resources section, not to mention SWAT's aforementioned excellent help links, should help you get started before we continue this series in my next column. Until then, be safe!

Resources

Christopher R. Hertel's On-Line Book *Implementing CIFS*, a Comprehensive Source of Information on All Things CIFS/SMB-Related: www.ubiqx.org/cifs

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

“The Official Samba 3.2.x HOWTO and Reference Guide”: us1.samba.org/samba/docs/man/Samba-HOWTO-Collection

Part II

Last month, I began a multipart series of articles on how to build a secure file server based on Samba for local (non-Internet-facing) use. I gave an overview of file server security goals, described why Samba might be the best tool for the particular job of serving “network drives” to clients on different platforms connected to a Local Area Network, defined a number of Samba acronyms and concepts, and explained how to install the Samba server daemons, client utilities and its configuration GUI, Swat.

This month, I expand upon our usage scenario and begin demonstrating how to construct an smb.conf file that executes this scenario in a secure fashion. As usual, I cover not only security, but also how to get things working in general—it isn't helpful to be told how to secure a process that isn't behaving the way you expect in the first place.

Usage Scenario

As I explained last month, we want to build a convenient and secure file server that supports both Windows and Linux (and other *nix) clients. Specifically, we want to build a non-Internet-facing Samba file service that supports several different levels of security: Guest (anonymous) access, read-only access for some authorized users and read/write access for other authorized users.

To use more Samba-specific terms, our server will operate with “user-mode” security, using a combination of local Linux/UNIX user account information and Samba-specific hashes of those users' passwords to authenticate access to workgroup resources. A workgroup, you may remember from last month, may include shares provided by multiple Samba servers, but each server in that workgroup must maintain its own user database independently (which is why Domains and Active Directories are better choices than workgroups for more complex environments). Accordingly, our sample workgroup will use a single server.

To flesh out our example scenario still further, let's suppose I've got a boardinghouse and my tenants are a trio of FBI special agents: Skippy, Knute and Pepe. Being fond of my cooking, they keep a close watch on my weekly meal schedule, which I post on my Samba server every Sunday night—you can bet nobody works late any evening on which tater tots will be served. This schedule is a public document, as far as I'm concerned (I'm vain about my cooking).

If my resident agents help mow the grass, feed the hedgehogs and tune the piano, they get a break on rent. So, I also maintain a schedule of chores they all can read, but which, of course, I don't want them to be able to change themselves (imagine the outrage if Pepe always got to feed the hedgehogs).

Besides, being secret agents after all, these guys don't want anyone else to know who'll be outside raking the compost on any given Saturday—you never know when the enemies of freedom might strike. So, the chore schedule is private, and it can be read but not altered by my tenants.

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

Finally, unbeknownst to the boys, their boss has asked me to log their Web-surfing activity from my firewall. Although the joke's on him (they all use TOR), these logs are nonetheless super-secret.

Actually, those logs probably don't belong on any file server at all, but sometimes I amuse myself by adding fake entries to Skippy's log ("GET HTTP://thesharperimage.com/expensive_gifts_for_your_boss.html"), so my firewall stores them on a restricted share on the Samba server.

To summarize, I need to create a workgroup (I'll call it FED-CENTRAL) with four user accounts (skippy, knute, pepe and mick) and three file shares (SUPPER, CHORES and BUZZ-OFF).

Samba Configuration: Global Settings

With that, we are ready to go. Assuming you successfully installed the Samba server and client packages per last month's instructions, the first step in configuring Samba is to set some global variables.

There are two different ways to configure Samba. The first is by editing `/etc/samba/smb.conf` directly using your text editor of choice, and then restarting the `smbd` and `nmbd` Samba daemons. You very well may gravitate to this method as soon as you're comfortable with Samba, because it's the quickest and most direct way to change Samba's behavior.

Lately, I've become a big fan of the second way, however: Swat, the Samba Web Administration Tool. If you're scandalized by my endorsing a graphical tool that requires you to set a root password (which, by default, doesn't exist on Ubuntu systems), see last month's column. Suffice it to say that in this case, I'm not talking about an Internet-facing system, and the educational benefits of Swat outweigh its security risks here.

Besides, Swat really isn't a crutch; it simply presents you with a Web form for assigning values to all possible variables in `smb.conf`, with convenient Help buttons that send you directly to the appropriate section of the relevant man page. The more you use Swat, the more comfortable you'll be editing `smb.conf` directly. How many GUIs can you say *that* about?

I'm going to assume you had no problems with the instructions I provided last month on installing Samba and Swat, including configuring and restarting `inetd`, and that Ubuntu users were able to stomach issuing the `sudo passwd root` to set a root password. (And, even if you weren't, or simply prefer not to use Swat, all of what follows still should be useful, because the variables and values in my Swat screenshots and examples are the same as those contained in `smb.conf`.)

Running Swat is easy. Simply start your browser of choice, and point it to `http://localhost:901/`. The first thing you'll see is Swat's Home page, which consists of a row of navigation buttons (Home, Globals, Shares, Printers and so forth). These appear on every one of Swat's screens, but unique to the Home page is a list of links to local man pages, HOWTOs and even complete books. I leave it to you to explore those; this page leads to a wealth of useful information for Samba users at all levels of skill and experience.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

For now, however, let's dive right into Samba's global settings. If you click the Globals button, and then scroll down to where the actual settings begin, you should see something like Figure 1.

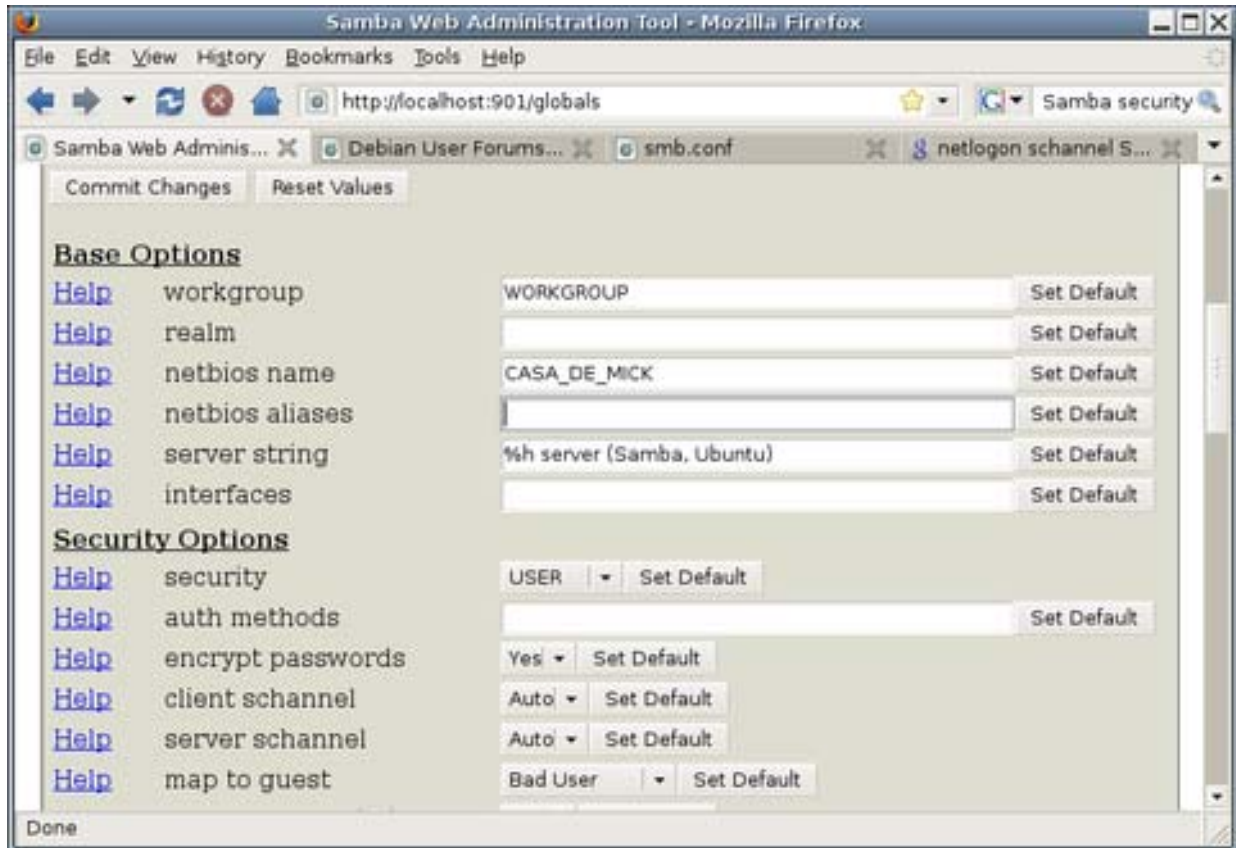


Figure 1
Some Global Settings

Obviously, we want to change the value of workgroup from WORKGROUP to FED-CENTRAL. The default for netbios name, however, is the hostname Samba automatically read in from /etc/hostname, and you usually can leave that alone, although you don't have to. This is the name that turns up in people's "network neighborhood" browser when they look for your server.

The default value for security, which is user, also is exactly what we want. The same is true of encrypt passwords being marked yes.

The next two variables, however, client schannel and server schannel, need to be changed. Schannel refers to the secure channel method of allowing Samba clients to log on to Samba servers, and we don't want this to be optional. We want it to be mandatory. Therefore, for both of those variables, we should change the value from auto to yes.

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

The last variable shown in Figure 1 is map to guest. That value tells Samba under what circumstances it should grant guest access to a client that has just had a failed login attempt. Samba's default for this is normally never, which effectively disables guest access. But, as you can see in Figure 1, on my Ubuntu system, the default value actually is Bad User, which means that if people try to log on with a nonexistent user name, they'll be given guest access.

If you scroll farther down on Samba's Global page, you should see something like Figure 2.

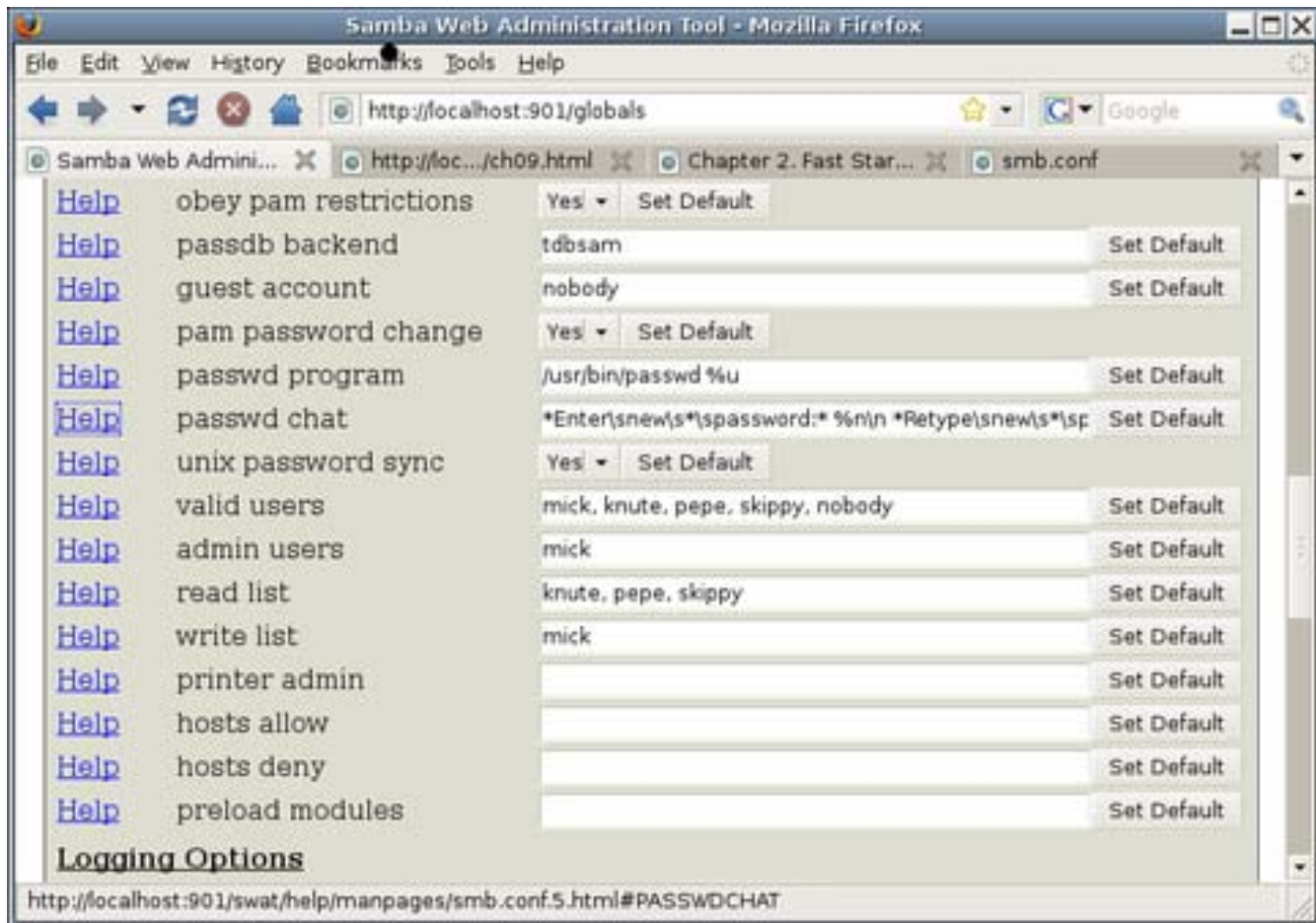


Figure 2
More Global Settings

Continuing through these global settings, obey pam restrictions implies that Samba will honor PAM (Pluggable Authentication Modules) settings. But in practice, if encrypt passwords remains set to yes, Samba will ignore PAM altogether.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

passdb backend specifies what type of database Samba should use to store its password hashes. The default (tbdsm) is usually the best choice.

guest account is the local Linux account that will be used for clients who fail authentication, as I described earlier when talking about map to guest. passwd program, passwd chat and unix password sync involve how and whether Samba mediates end users' attempts to change their passwords via Samba (Windows file sharing) sessions. Leave these at the default settings unless you don't want users to be able to change their passwords that way.

By now, you may be wondering, what's the difference between Samba's password database and the list of hashes stored in /etc/shadow, given the fact that they correspond to the same set of local user accounts? The short answer is, Samba (SMB/CIFS) uses an authentication protocol with which UNIX password hashes are not compatible.

The bad news is that Samba's password database is, thus, totally redundant with Linux's, and it creates the potential for users having to remember two different passwords. The good news is that if passwd program and passwd chat are set correctly (which they should be by default, if you use your Linux distribution's official Samba packages), and unix password sync is set to yes, Samba automatically will update users' Linux passwords every time they change their Samba password. (I talk about this more in the next section.)

Moving on, valid users allows you to specify a list of Linux/UNIX user accounts to which you want to grant access to Samba shares. The default value "" (null) results in all local Linux accounts being valid. For our example scenario, I've set valid users to mick, knute, pepe, skippy and nobody.

admin users allows you to grant superuser privileges on all shares for one or more local user accounts, regardless of Samba or Linux file permissions on that share. Be careful with this setting! It has the effect of executing local commands as root on behalf of such users. In Figure 2, I've specified mick as an admin user, because I often use that account for system administration tasks anyhow.

read list allows you to specify which users should have default read-only permissions on shares. As you can see in Figure 2, I've set our read list to knute, pepe and skippy.

Similarly, write list specifies a list of users who should have read-write privileges by default. I've set that value to mick.

printer admin is out of the scope of this article for now (though I may cover printer shares later in this series). hosts allow and hosts deny, however, are noteworthy. They allow you to create TCP Wrappers-style access control lists. hosts allow is a whitelist of IP addresses, network addresses, hostnames or domain names that should be allowed to connect by default (assuming successful authentication, of course).

hosts deny is a blacklist, also consisting of IP addresses, network addresses and so forth, whose members won't even be permitted to attempt authentication. Samba will break any connection

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

attempted by any host matching this list. The `hosts_access(5)` man page provides complete information about the syntax of the values of these two variables.

And, that's it for global settings, for now. To write the changes we've made to our working `/etc/samba/samba.conf` file, click Swat's Commit Changes button.

Some of the variables you set to custom strings, such as valid users, may not appear when the screen refreshes. To see them, simply click the Advanced View button (next to Change View To: near the top of the page).

Setting Up User Accounts

The last task we've got space for this month is setting up our user accounts, and there are four steps:

1. Create the accounts under Linux.
2. Assign those accounts Linux passwords.
3. Create Samba password database entries for each.
4. Have the users change their Samba passwords.

Step one is to use whatever method you usually use to create user accounts on your system—either by using your system administration GUI of choice (such as GNOME's Users and Groups applet) or via the commands `useradd`, `userdel` and so forth.

For example, to create Pepe's account, I could use the following command. Note the `sudo`, necessary for Ubuntu. On other distributions, `su` to root before executing these commands, and omit the `sudo` that each begins with here:

```
bash-$ sudo useradd -c "Pepe" -m -g users pepe
```

This creates the user account `pepe` with the comment `Pepe`, automatically creates a home directory (`/home/pepe`) and assigns it to the group `users`. To be extra paranoid, you could insert the string `-s /bin/false` after `-g users`, which will disable normal Linux logins for Pepe's account, making it useless for anything other than Samba access.

Step two is to set each user's Linux password, like this:

```
bash-$ sudo passwd pepe
```

Obviously, you need to communicate whatever password you set here to Pepe in a secure fashion, and Pepe will need to change this password to something you don't know. (But that part happens in step four.)

Step three is to use the `smbpasswd` command to create each user's Samba password database entry, like so:

```
bash-$ sudo smbpasswd -a pepe
```

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

You'll be prompted to set and confirm Pepe's Samba password, after which the new account will be added. It's probably a good idea to use the same initial password here that you used in step two.

Finally, you'll want Pepe to log in to the system (assuming you *didn't* set his shell to `/bin/false`) and issue the following command:

```
pepe@casademick$ smbpasswd
```

Pepe will be prompted for his old password, his new password and confirmation of his new password. Assuming all three of those are good, Samba will change both Pepe's Samba password and his Linux password accordingly. Note that this synchronization does *not* occur when you create a new Samba password entry as root, using the `-a` flag.

If Pepe has an invalid shell, such as `/bin/false`, you'll have to let him sit at your console while you type the command `sudo smbpasswd pepe`, and then turn your back while he changes his password. You'll then need to do the same thing with the command `sudo passwd pepe`, because Samba does not synchronize Linux/UNIX passwords if you execute `smbpasswd` as root.

Regenerating smb.conf in Debian/Ubuntu

What if, in the process of tinkering with your Samba configuration, you so completely lose track of what you've changed versus what you started with that you want to begin again with the default `/etc/samba/smb.conf` file? And, what if you failed to create a backup copy of `smb.conf` before you changed it?

You might think Swat could do this. Swat has default buttons next to each configuration option. Clicking a default button is supposed to replace your custom value with the value from the default `smb.conf` file included with Samba. However, in my own experience, the behavior of these buttons is erratic. Sometimes null values are (incorrectly) returned, and clicking the default button for every option is time consuming anyhow.

My advice is that if you're using Debian or one of its derivatives, such as Ubuntu, and you need a fresh `smb.conf` file, you should completely un-install the package `samba-common`, and then re-install it. (This also will result in things that depend on `samba-common` to be un-installed, so note which packages you'll need to re-install after you've restored `samba-common`.)

In between removing and re-installing `samba-common`, you may want to check `/etc/samba` to make sure `smb.conf` is truly gone, and delete it if it isn't.

Conclusion

We've specified our usage scenario, set up some basic global settings using Swat and started adding users. Next month, we'll create the actual shares, but if you can't wait until then, you'll have no problem figuring out how using Swat's ample documentation. The "Official Samba 3.2.x HOWTO and Reference Guide" (see Resources) also may help. Have fun, and be safe!

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

Resources

Christopher R. Hertel's On-line Book *Implementing CIFS*, a Comprehensive Source of Information on All Things CIFS/SMB-Related: www.ubiqx.org/cifs

"The Official Samba 3.2.x HOWTO and Reference Guide": us1.samba.org/samba/docs/man/Samba-HOWTO-Collection

Part III

This month, we continue our exercise in building a secure file server for our local LAN using Samba. In case you missed the first two installments, this is a non-Internet-accessible file server to which users of a LAN can mount virtual disk volumes.

The example scenario I'm using is a boarding house in which I need to provide a world-readable file share containing menus (SUPPER), a group-readable share containing schedules of chores (CHORES) and a private share containing copies of Web logs (BUZZ-OFF).

Last month, we used Samba's Swat tool to configure our Samba server's Global settings. We then created four user accounts: mick, knute, pepe and skippy. Mick, of course, is me. Knute, Pepe and Skippy are the three FBI agents who rent my rooms and who are interested in my daily menus and weekly schedules of chores, but with whom I'd rather not share my Web logs.

This month, we create a public share for menus called SUPPER and a nonpublic but group-readable share for chore lists called CHORES. (We'll save the private share, BUZZ-OFF, for next time.)

Creating a World-Readable File Share

As we've seen, Swat is arguably the best tool for configuring `smb.conf`, Samba's primary configuration file. Other tasks, like creating new user accounts, are best done from a command line (last month, we used the standard commands `useradd` and `passwd` to set up our accounts under Linux, and then `smbpasswd` to create corresponding Samba accounts).

To create shares, however, we can return to Swat. Unsurprisingly, the navigation button you must click is labeled Shares. After you do that, type the name SUPPER in the box to the right of the Create Share button, and then click that button. You should see something like Figure 1.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

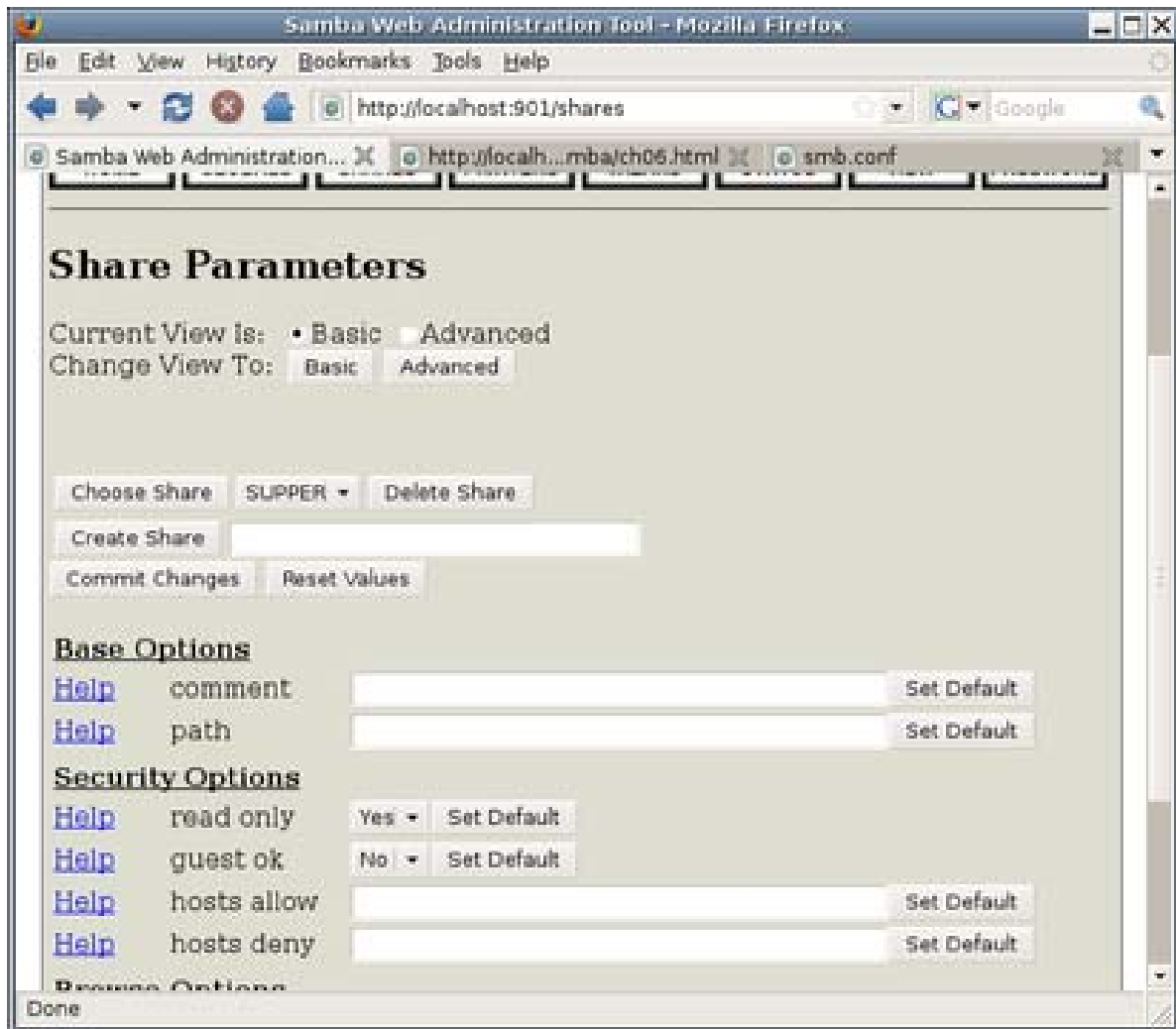


Figure 1
Creating a New File Share

Under Base Options, I set comment to Mick's Menus. Then, I set path to /home/mick/supper. This will be our weekly menu folder.

The value of path has to correspond to a real directory on your server. Furthermore, the Linux permissions and ownership of this directory need to be set to allow the desired level of access you want to grant. In this example, the directory listing of /home/mick/supper looks like this:

```
drwxr-xr-x 2 mick users 4096 2008-09-12 01:44 supper/
```

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

As you can see, the user mick has read-write-execute permissions, but group and other have only read-execute permissions. Now isn't the time for a primer on filesystem security (actually I've already written one: "Linux Filesystem Security", in the October and November 2004 issues of *Linux Journal*). Suffice it to say for now that the commands for creating directories, setting user and group ownership and setting permissions, respectively, are `mkdir`, `chown`, `chgrp` and `chmod`.

A Note on Figures 1 and 2

The screenshots in Figures 1 and 2 show Ubuntu's default values for the various settings in Swat. They, therefore, do *not* provide, all by themselves, a model of how to configure Samba securely! Read the accompanying text for my recommended (secure) settings.

Let's set some security options shown in Figure 1. By default, at least on Ubuntu systems, Swat displays only four options under this section in its basic view, but that's a reasonable starting point.

The first of these is read only, which I leave at the Ubuntu default of yes, even though I want the user account mick to be able to publish new menus. (The setting write list, which I'll describe a little later in this article will override this setting.)

The second security setting shown in Figure 1 is guest ok, which I change to yes. (My guests, and those of my boarders, certainly will be keenly interested to know what side dishes will accompany Tuesday night's Coconut Tater-Tot Casserole.)

I should pause here for a quick review of how guest access works in Samba. Last month, when we configured Samba's global settings, we set the option `map to guest` to `Bad User`, which caused Samba to treat clients who log in with nonexistent user names as guests. We set the option `guest account` to `nobody`, which means that when people log on as a guest (either by providing a bad user name or by actually logging in as `nobody`), they will be logged in under the account `nobody`.

None of these global settings has any effect on a given share unless that share's `guest ok` option is set to yes. As we'll see shortly, that doesn't actually give guests any *permissions* on that share unless we do just a little more work.

First, there are two more security options to attend to in Figure 1: `hosts allow` and `hosts deny` can be used to define TCP Wrappers-like, network-level access controls on your share. You can learn everything you need to know about this from the `hosts_access(5)` man page.

In Figure 1, `hosts allow` will be set to `192.168.44.`, which means "allow access from clients whose source IP address' first three octets are 192.168.44". In our example scenario, this corresponds to my local LAN address of `192.168.44.0/24`. `hosts deny` is set to `ALL`, which means "deny access to all clients who do not match any value in `hosts allow`."

In my opinion, there's no good reason not to use `hosts allow` and `hosts deny` with Samba unless your LAN is very complicated. It's not as important as making proper use of user and group accounts, enforcing the use of strong passwords and other things you should be doing, but it's nonetheless a useful layer in our defense onion.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

At this point you may be wondering, how do we tell Samba who has write access and who has read-only access for this share? The four security options we've covered don't address that. The answer is, we've already established some default settings for this in the global section, and share-specific authorization controls can be set by switching from basic to advanced view in Swat, by clicking the Advanced button near the top of the screen. When you do that, you'll see something like Figure 2.

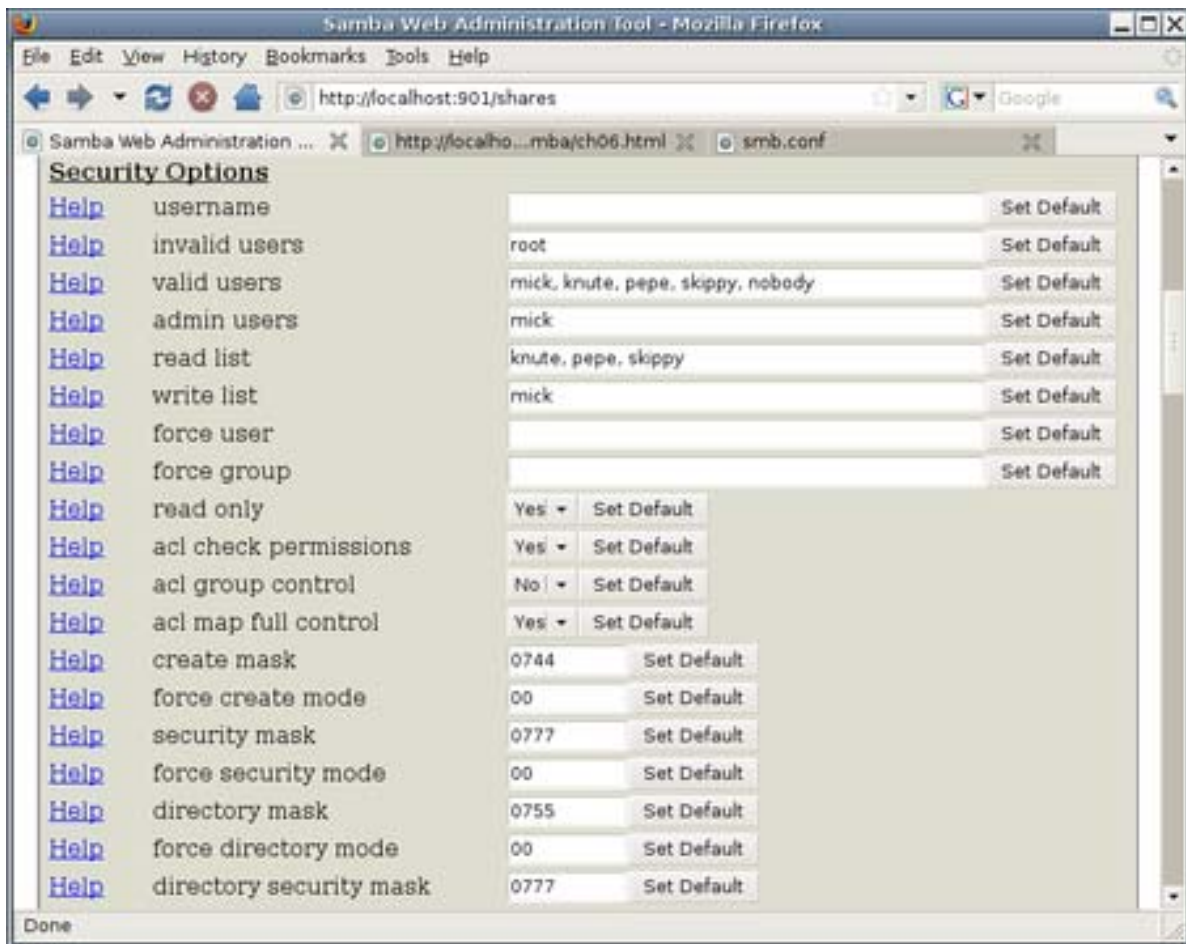


Figure 2.
Share Security Options in Advanced View

But wait, what's this? Where did those values for valid users, read list and so forth come from, given my earlier sidebar note about these screenshots showing default settings?

As it happens, many of Samba's options can be declared *both* as global settings and as share-specific settings. When you set up a new share, Swat copies the values of any such options you set up under

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

the global settings to the new share. So, Figure 2 represents Swat's settings after I've set up the global section but before I've fine-tuned the SUPPER share.

And, I do need to fine-tune it! On the one hand, invalid users is set to root as in the corresponding global option, which is a good value to propagate here; it's never a good idea to log in to much of anything directly as root.

But because I want this to be a public share, I'm going to remove all the users listed in valid users, which will have the effect of allowing clients to log in using *any* user name they provide. (Remember, though, anyone logging in with a user name outside the Samba user database or /etc/passwd will be logged on as nobody—that is, as a guest.)

Similarly, I'm going to empty read list as well, as read only is set to yes anyhow. (read list is sort of a blacklist: anyone whose user name is listed here will be granted only read access to this share regardless of *any other setting* in this share or under Globals.)

Another setting I'm going to empty is admin users. Like I said last month, this is a dangerous setting, and it's usually unnecessary. (I really shouldn't have set it to mick in the global section!) Not only will admin users operate with full Linux root privileges, all files they create will have a user owner of root, which can complicate both Samba and Linux filesystem permissions. Most of the time you might be tempted to set this option, it's probably sufficient instead simply to give that user write access.

And, you can do that with the option write list. In this case, we can leave the value of mick inherited from Globals.

The last security setting to change is create mask. This option determines the UNIX permissions that will be given to any files moved into or created in the share. Its value must be a chmod-style octal mode, as described in the chmod(1) man page.

The default value 0744, shown in Figure 2, translates to “owner read+write+execute, group read, other read”. However, because this share is going to contain text files, there's no reason for the group-execute bit to be set; 0644 (owner read+write, group read, other read) is a better choice.

To review, and for clarity's sake, Figure 3 shows the changed settings for these security options in Swat's advanced view.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

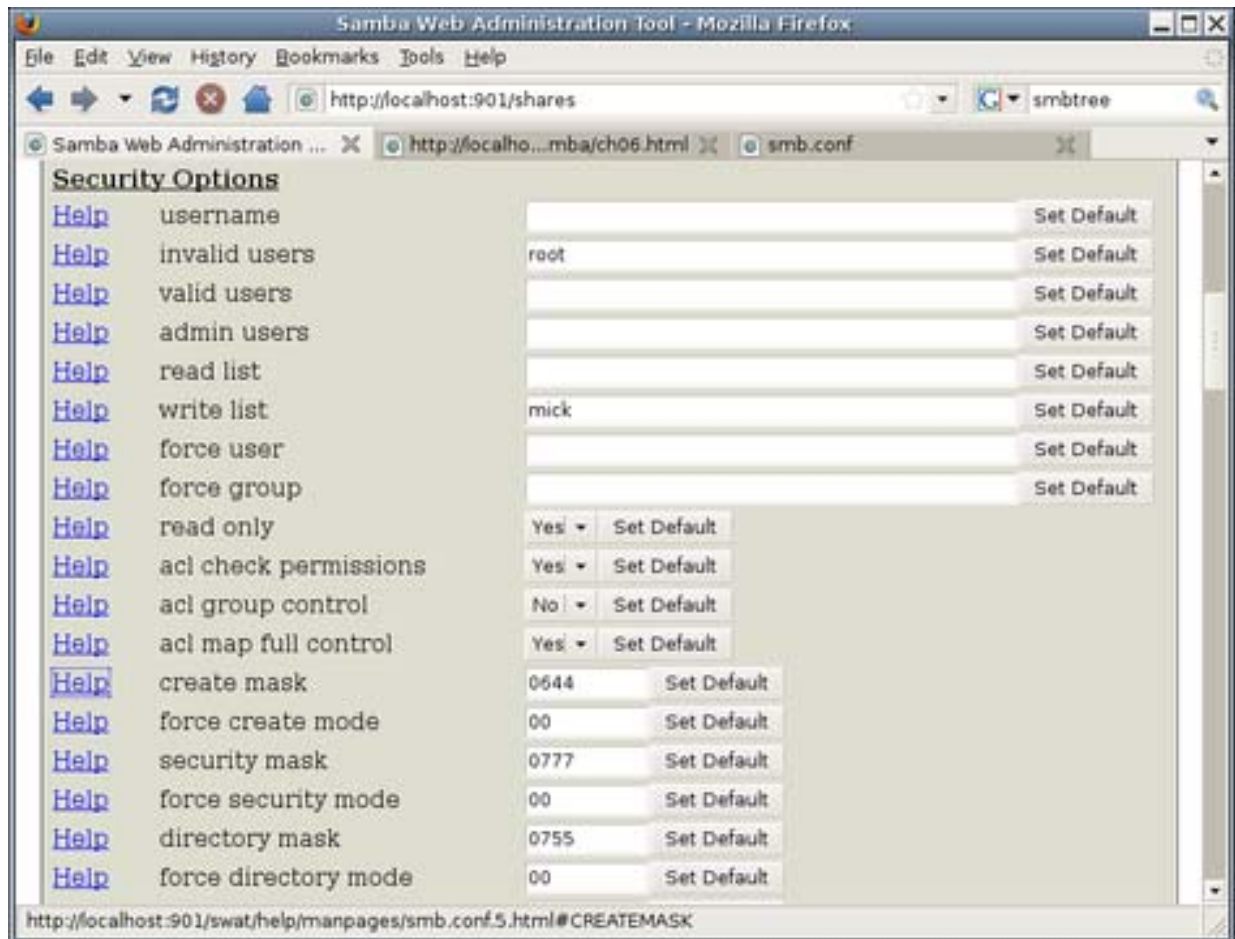


Figure 3
New Share Security Settings

We're almost done configuring this share. There are just two more options to check, and now you can switch back to basic view to find them quickly. The Browse Option browseable is set to yes by default on Ubuntu systems, which is appropriate for a public share.

The EventLog Option available, on the other hand, which is used to enable or disable a share, has the rather sensible default value of no. I say sensible, because it's never a good idea to activate anything before you're finished configuring and securing it! But, we are in fact done securing this share, so we'll change available to yes.

The last step is to click the Commit Changes button near the top of the Swat page. On my system, any time I click this button, the view resets to what appear to be default settings for printer shares! If this happens on your system too, all you need to do is click the Choose Share button again to display the changes you just committed.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

After you create, delete or reconfigure a share, the changes will be applied immediately to your running Samba daemons; there's no need to restart any of them.

Testing Samba Shares

Now that the SUPPER share is configured and available, it should start showing up in the Network Neighborhood (or other Windows network browser) of users connected to the LAN. Your Samba server, which we've configured to be a Browse Master for its workgroup, achieves this by sending out broadcasts.

However, in my experience, network browsers are often unreliable—it can take a while for your new workgroup, servers and shares to show up, and sometimes things disappear for no apparent reason. (Even for Windows clients, using the Map Network Drive feature to specify your share's path is both faster and more reliable than using the Network Neighborhood browser.)

So although you might get decent results testing your new share by simply firing up a network browser, I recommend using Samba's command-line tools instead, namely, smbclient and smbtree, which are included in Debian and Ubuntu's smbclient package, and in Red Hat and SUSE's samba-client package. I'll leave it to you to explore the smbtree(1) and smbclient(1) man pages, but I will give you a couple usage examples.

smbtree is a text-based Windows network browser that sometimes performs better than GUI-based browsers. To view all available workgroups, servers and public shares on your local LAN, use this command:

```
bash-$ smbtree -N -b
```

smbclient is a much more versatile command that can be used both to view and use Samba shares. To use smbclient to connect to our new share as the user nobody (guest), you can type:

```
bash-$ smbclient //CASA_DE_MICK/SUPPER -U nobody
```

Note the share-name syntax: //<servername>/<sharename>. You can use an IP address instead of the actual server name; this can result in a quicker login, because it allows smbclient to skip the name-resolution step. (Have I mentioned lately how inefficient the SMB/CIFS protocol is?)

Note also that to test the Bad User (guest-failover) behavior I described earlier, this command should be functionally equivalent to the previous one:

```
bash-$ smbclient //CASA_DE_MICK/SUPPER -U totallyfakeusername
```

You'll be prompted for a password. Simply press Enter without typing one (your nobody account shouldn't have a password!). If everything is working, you should see something like this:

```
Anonymous login successful
Domain=[FED-CENTRAL] OS=[Unix] Server=[Samba 3.0.28a]
Revised February 1, 2009
```

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

```
smb: \>
```

At this point, you now have the Samba equivalent of an FTP shell—in fact, this environment is designed to be similar to FTP clients. To see a list of all available commands, you can enter ? or help. For now, we'll just do a quick directory by entering dir:

```
smb: \> dir
.                D      0  Tue Oct  7 13:22:28 2008
..              D      0  Tue Oct  7 13:21:16 2008
0-mon_filetmingon.txt      51  Mon Oct  6 21:05:34 2008
1-tues_gruel.txt          47  Tue Oct  7 13:05:54 2008
2-wed_beefmushcasserole.txt  5  Tue Oct  7 13:06:32 2008
```

```
52008 blocks of size 262144. 13782 blocks available
```

I'll leave it to you to figure out how to test copying files in both directions (put should work only for the user mick, but everyone else, including guests, should be able to list, get and read files).

Creating a Group-Readable File Share

On the strength of our SUPPER-creating experience, you'll find it fast and easy to create the group-readable share CHORES (which will contain lists of household tasks my boarders can perform in exchange for a rent discount). This share will be very similar to SUPPER: mick will have read and write access; pepe, skippy and knute will have read access only. However, unlike SUPPER, guest access will not be permitted.

Accordingly, after typing a new share name (CHORES) into the Create Share field and then clicking the Create Share button, we'll need to be sure to leave guest ok set to its default value of no. We'll set comment and path to Chore lists and /home/mick/chores, respectively (having first created this directory in a terminal window, and setting its ownership and permissions to be the same as for /home/mick/supper).

hosts allow and hosts deny can be the same as for SUPPER. browseable can be left at yes, but available should be left at no for now.

Figure 4 shows these settings (except available) for our new CHORES share.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

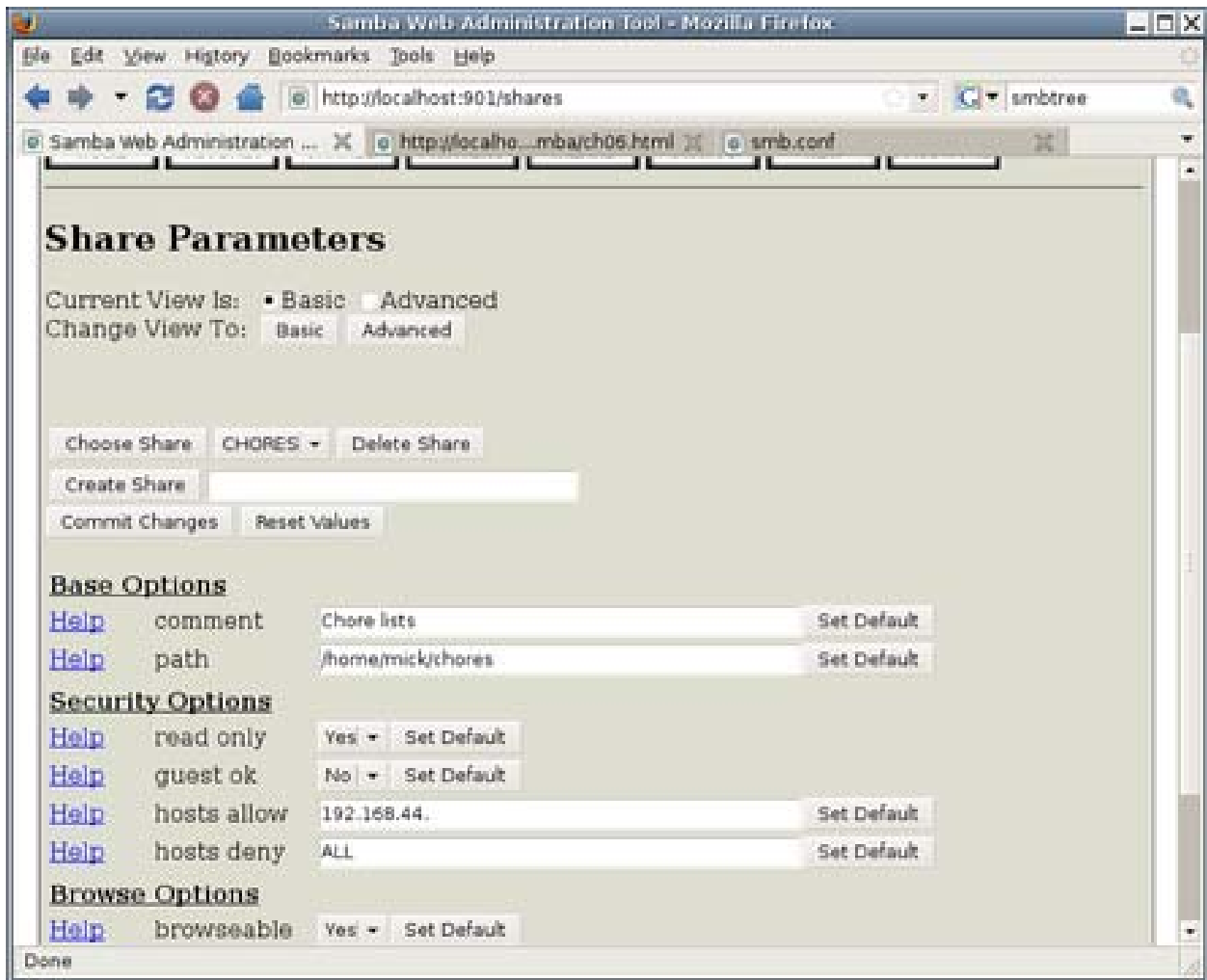


Figure 4
Basic View Settings (Customized) for CHORES

Now, we'll switch to Swat's advanced view for this share (if you aren't there already) by clicking the Advanced button. As with SUPPER, we'll blank out admin users, because we're paranoid, and also read users, as read only already is set to yes.

As you can see in Figure 5, however, I'm employing a bit of useful laziness in the valid users field for CHORES.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

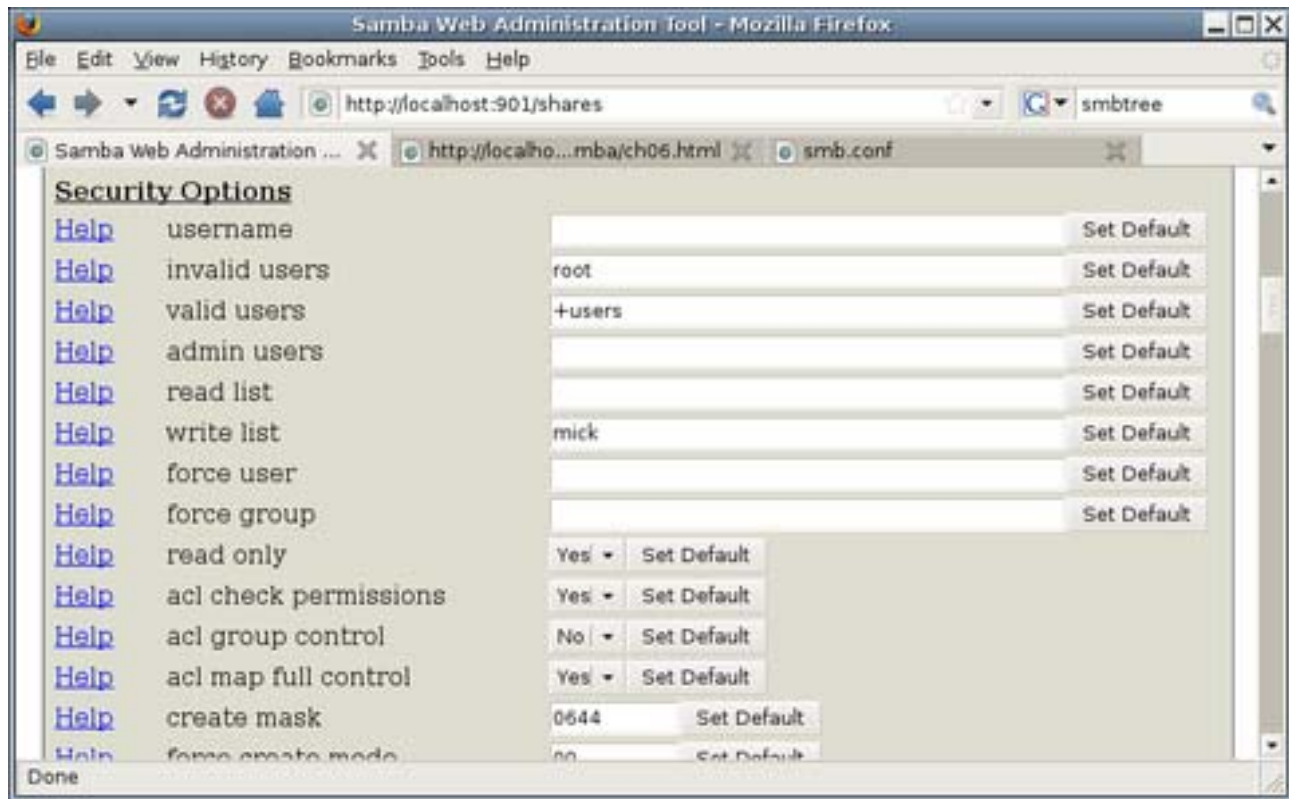


Figure 5
Advanced Security Settings (Customized) for CHORES

In the valid users field in Figure 5, the + in front of users instructs Samba to look up the name users in /etc/group, and then replace this entire value with a list of all members of the group users. Because on this server that group consists of mick, knute, pepe and skippy, Samba ultimately will set the value of valid users to mick, knute, pepe, skippy.

Needless to say, be careful with group names in this context. Before using one in Swat (or directly in smb.conf), be sure you know for certain exactly which user accounts belong to that group.

The quickest way to do this is to look up the group name in /etc/group and note its numeric value, noting also any secondary group members it has. Then, see which users in /etc/passwd have that group's number listed as its primary group.

Here's how this looks when enumerating the group users on my Ubuntu system:

```
mick@ubuntu@:~$ grep users /etc/group
```

```
users:x:100:
```

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

```
mick@ubuntu:~$ grep :100: /etc/passwd

dhcp:x:100:101:~/nonexistent:/bin/false
mick:x:1003:100:Mick Bauer:/home/mick:/bin/sh
knute:x:1004:100:Knute:/home/knute:/bin/sh
pepe:x:1005:100:Pepe:/home/pepe:/bin/sh
skippy:x:1006:100:Skippy:/home/skippy:/bin/sh
```

As you can see, there are no secondary users listed at the end of the user's entry in `/etc/group`. My second `grep` command turned up five users, not the four I was expecting, but `dhcp` matched only because its numeric user ID (not its group ID) is 100.

The other settings we should change are `create mask`, which we'll again set to `0644`, and then `browseable`, which we now can safely change to `yes`. Finally, we can click the `Commit Changes` button, and `CHORES` is ready to go. Preferably using another system, test it to make sure it works the way you expect.

Conclusion

That's all we've got space for this month. Next time, we'll create that third, mick-only share (I'll bet you can figure that out yourself beforehand), create persistent Samba mounts on our client systems using `smbmount` and at least briefly address some miscellaneous Samba security topics, such as how to make Samba automatically and safely serve people's home directories. Until then, be safe!

Resources

"Linux Filesystem Security, Part I": www.linuxjournal.com/article/7667

"Linux Filesystem Security, Part II": www.linuxjournal.com/article/7727

Samba Security Part IV February 1, 2009

For the past four months in this column, we've been building a secured Samba server for our local LAN, using `Swat`. To spare those of you who have been following this series a fourth summary of the usage scenario, let's suffice it to say, we're creating a series of file shares with varying user permissions.

This month, I wrap up the series by showing how to create a restricted, "owner-only" share and how to use `mount.cifs` to make persistent Samba mounts on your client systems.

What We've Done So Far

Last month, we created a public share, `SUPPER`, and a nonpublic and group-readable share called `CHORES`. Prior to that, we had set up some global variables that are inherited by all shares. Those global variables were:

```
workgroup          = FED-CENTRAL
security           = user
client schannel    = yes
```

Revised February 1, 2009

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

```
server schannel      = yes
map to guest         = Bad User
guest account        = nobody
unix password sync   = yes
valid users           = mick, knute, pepe, skippy, nobody
read list            = knute, pepe, skippy
write list           = mick
```

Attentive readers of Part II of this series [December 2008] may notice that I omitted “admin users” here, even though in Part II, I had set that to mick. This was an embarrassing mistake. On Ubuntu systems at least, this wreaks havoc with how Samba interacts with Linux file permissions.

You'll recall that setting “admin users” causes listed users to be logged on to Samba as root after successfully authenticating as themselves. In other words, if “admin users” is set to mick, any time mick successfully logs on to any share, he'll actually be logged on as root. The *expected* result is that mick, therefore, will have superuser privileges and won't be restricted from doing anything at all. In practice, the results tend to be much less predictable than that.

For example, on my Ubuntu 8.04 system, suppose I set “admin users” to mick, create a directory on the underlying Linux filesystem that's owned by mick and has permissions of -rwx----- (or 0700), and then I create a Samba share mapped to that directory that has no guest access or read access (that is, a share like the one I'm about to show you how to set up).

If I then try to connect to this share with this command:

```
bash-$ smbclient //CASA_DE_MICK/BUZZ-OFF -U mick
```

and enter the correct password when prompted, sure enough, silently and behind the scenes, I'll actually be logged on as root. But the result of this login will be:

```
Domain=[CASA_DE_MICK] OS=[Unix] Server=[Samba 3.0.28a]
tree connect failed: NT_STATUS_ACCESS_DENIED
```

What? How can this be? Access shouldn't be denied to anything, for root, should it? But denied it will be, if the share in question maps to a directory not owned by root. This may or may not happen on non-Ubuntu systems. My point is that using the “admin users” parameter may result in unpredictable interactions between Samba and Linux filesystems.

As I said last month, letting people use Samba shares with root privileges is dangerous anyhow. Samba client software isn't the correct tool for Samba system administration, Swat is. So now we have two good reasons always to leave “admin users” empty!

Now, let's move on to our share-specific settings. The smb.conf variables that configured SUPPER, as set via the Swat tool, looked like this:

```
path                = /home/mick/supper
read only           = yes
Revised February 1, 2009
```

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

```
guest ok      = yes
invalid user  = root
valid users   =
read list    =
write list    = mick
admin users   =
hosts allow   = 192.168.44
hosts deny    = ALL
create mask   = 0644
browseable    = yes
available     = yes
```

These variables are set nearly the same for CHORES, except:

```
path          = /home/mick/chores
guest ok      = no
valid users   = +users
```

What do all these variables mean? I explained them in gory detail in the last three issues of *LJ*, and definitive descriptions can be found for all in the `smb.conf(5)` man page. Some of these will come into play this month too, as we create that restricted share.

Creating a Restricted Share

You'll recall that our Samba server has four user accounts: pepe, skippy, knute and mick, which correspond to my three roommates and me. These are UNIX user accounts on my Samba server's underlying OS, with corresponding but separate entries in the Samba server's separate user database. (I explained how to create and synchronize Samba user accounts in Part II of this series, in the December 2008 issue.)

For our restricted share, BUZZ-OFF, only mick should have read access *or* write access. No other user should have any rights at all on this share. Accordingly, when we create the directory to which this share will point, we'll be sure it's owned by mick and has a permissions mask of 0700 (u+rwx,g-rwx,o-rwx), like this:

```
drwx----- 2 mick users 4096 2008-11-04 00:00 buzz-off
```

Figure 1 shows the first round of parameters we'll set upon creating this share in Swat's Basic View.

Samba Security
By Mick Bauer
(Reprinted From the Linux Journal)

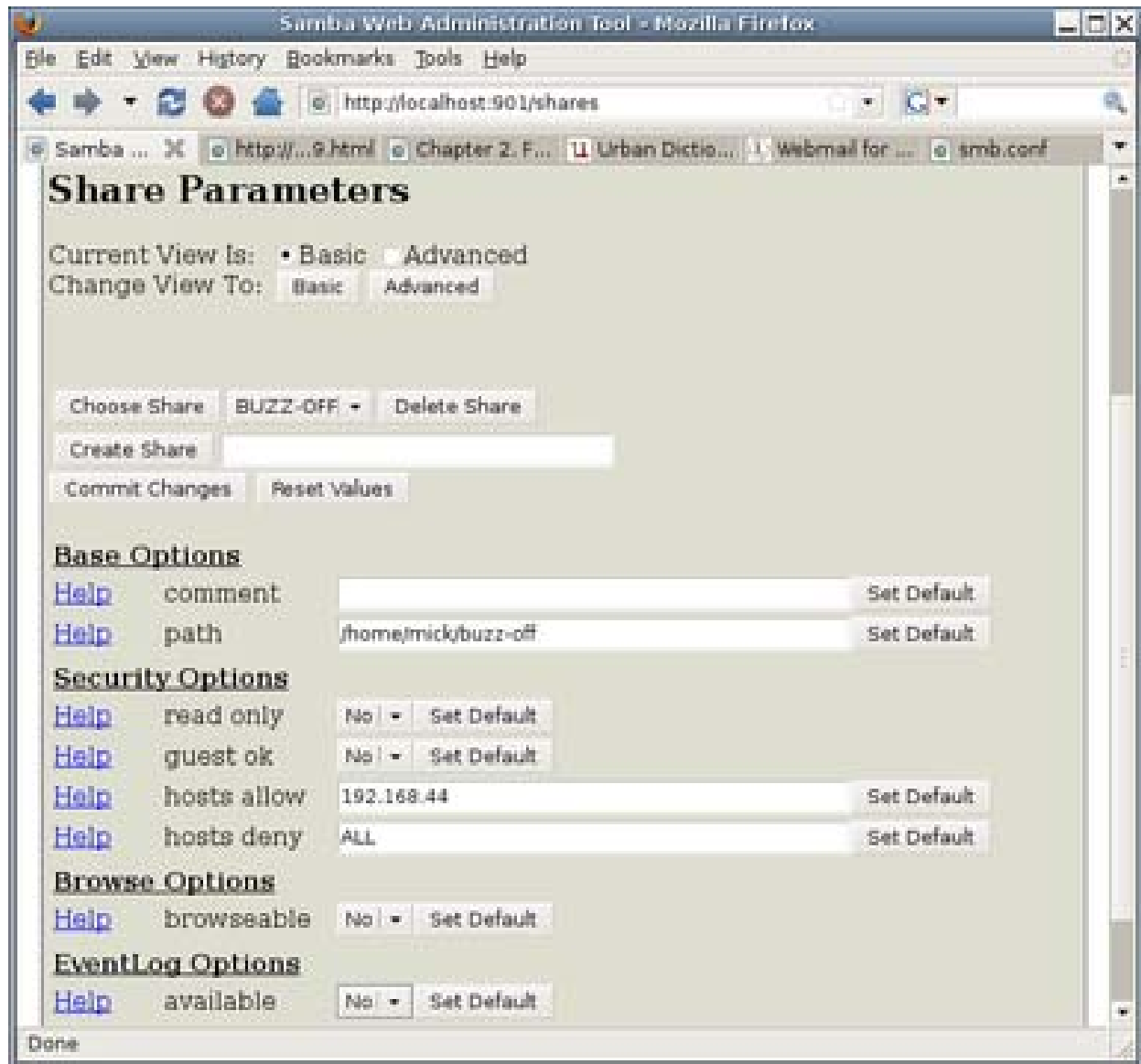


Figure 1
Restricted Share, Basic Settings

After setting the path, we set read only to no—I'll be creating new files in this share—and guest ok to no as well, because we don't want to allow any anonymous access. We'll set hosts allow and hosts deny the same as our other shares—to permit access only from the local LAN (your network address is, obviously, probably different).

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

We'll set browseable to no, so this share won't turn up in people's Network Neighborhood or in smbtree listings. To connect to this share, therefore, we'll need to specify its path when mapping it to a drive or connecting to it with smbclient.

And, we'll leave available set at no until we've clicked the Commit Changes button, clicked the Change View to Advanced button and changed some things in the Advanced View (Figure 2).

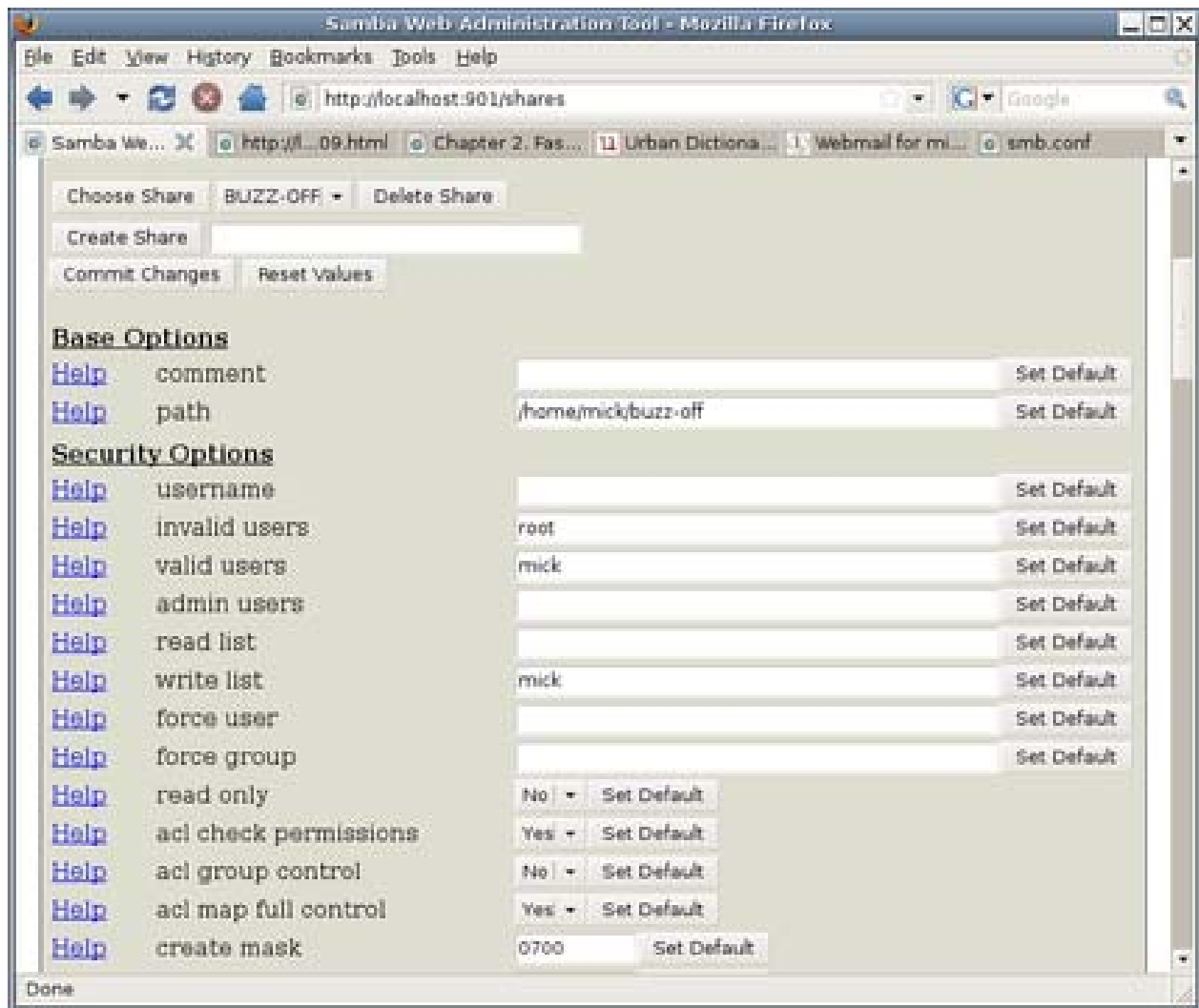


Figure 2
Restricted Share, Advanced Settings

As you can see, we're going to blank out the list of valid users except for mick and completely empty the contents of read list. The write list, however, correctly contains the single value of mick.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

The only other setting we need to change is create mask, which we'll set to 0600. This is different from the 0700 mask we used when creating the directory itself; the directory's execute bit needs to be set so the directory can be used, but the *contents* of this directory, which is what the share represents, do not.

Now we can change available to yes and click the Commit Changes button. Our restricted share is ready for use!

To test this, let's first make sure the share doesn't turn up in the local Samba browse list. We can perform this test using smbtree, like so:

```
bash-$ smbtree -N -b
FED-CENTRAL
  \\CASA_DE_MICK   iwazaru-ubuntu server (Samba, Ubuntu)
    \\CASA_DE_MICK\print$   Printer Drivers
    \\CASA_DE_MICK\SUPPER   Mick's menus
    \\CASA_DE_MICK\CHORES   Chore lists
    \\CASA_DE_MICK\IPC$     IPC Service (iwazaru-ubuntu server (Samba, Ubuntu))
```

Sure enough, the new share BUZZ-OFF doesn't appear in the browse list. But, is it nonetheless usable by mick, its owner? Let's find out with smbclient:

```
bash-$ smbclient //CASA_DE_MICK/BUZZ-OFF -U mick
Password: *****
Domain=[CASA_DE_MICK] OS=[Unix] Server=[Samba 3.0.28a]
smb: \>
```

It worked. I've got a Samba prompt! There's no reason not to try a quick directory listing before exiting:

```
smb: \> dir
.                D            0   Tue Nov  4 23:17:34 2008
..               D            0   Tue Nov  4 23:17:34 2008
access-log_10312008.txt      665   Tue Nov  4 23:17:34 2008

      52008 blocks of size 262144. 13229 blocks available
smb: \> exit
```

Everything worked as expected. One last test—just to be sure, I want to try logging in to the share as a guest user. Remember that our Samba server is set up to treat any login involving a nonexistent user name as a guest login:

```
bash-$ smbclient //CASA_DE_MICK/BUZZ-OFF -U totallyfakeuser
Password: *****
Domain=[CASA_DE_MICK] OS=[Unix] Server=[Samba 3.0.28a]
tree connect failed: NT_STATUS_ACCESS_DENIED
```

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

It failed, in the expected and appropriate way. Our restricted share is accessible, insofar as we want it to be.

Using mount.cifs for Persistent Samba Mounts

Now that I've got a restricted share available to me, suppose it will contain things I need to read and change on a regular basis. Do I need to access it via an interactive smbclient shell every time?

Of course not. The ability to mount remote Samba shares as though they were local volumes is one of the best things about Samba. You can do this by using the standard mount command, along with Samba's mount.cifs module, on your Samba client systems.

On Red Hat-derived and SUSE systems, the cifs filesystem and associated utilities are included with the standard samba-client package. On Debian, Ubuntu and other Debian derivatives, however, you'll need to install the package smbfs.

Although based on the same protocols, smbfs and cifs are actually two different things. cifs is newer than smbfs, and the smbmount command formerly used for mounting Samba file shares via the smbfs filesystem has been deprecated by the Samba team in favor of cifs and the mount.cifs module. smbfs and smbmount are still distributed with Samba, but they are not being actively maintained.

While we're installing Ubuntu packages, you'll also want the package winbind, which mount.cifs needs in order to resolve NetBIOS or Windows NT names (the Samba server we've been setting up uses NetBIOS name resolution, not Windows NT). SUSE users will need the package samba-winbind. I'm not positive, but I believe winbind is included in Red Hat/CentOS/Fedora's samba-client package.

After installing winbind, you should add the string wins to the hosts: line in /etc/nfsswitch.conf (only root can do this; you'll need to use su or sudo).

After mount.cifs and winbind are in place, you're ready to start mounting Samba shares. To do this manually from a command line, you can invoke the mount command as root or, as shown here, using sudo:

```
myclientlaptop-$ sudo mount -t cifs //CASA_DE_MICK/BUZZ-OFF  
/mymountpoint -o rw,suid,user=mick
```

In this example, we're telling mount to use a filesystem type (-t) of cifs. We're mounting, obviously, the share BUZZ-OFF on the server //CASA_DE_MICK, using the mountpoint ./mymountpoint (which is an existing directory within my current working directory). Note that I can, if necessary, use my Samba server's IP address rather than its NetBIOS (or Windows NT) name, in which case, that part of the command would look something like //192.168.44.123/BUZZ-OFF.

The -o gives a list of options for this mount. The first option, rw, lets me both read files from and write them to the share. suid causes any set-uid bits on files in the share to be acknowledged. user passes my Samba user name to the mount.cifs module so it can authenticate the session. After entering the above command, I'll be prompted first for the root password and then for mick's password.

Samba Security

By Mick Bauer
(Reprinted From the Linux Journal)

Whatever you do, do not enter your password on the command line using the `password=` option. Because shell commands may be logged in various places and are stored in shell histories, it's generally a terrible idea to use any password as a command argument.

If your Samba credentials are unimportant, for example, because they do not correspond to any user account with actual shell access, it's still a good idea to avoid passing its password to a command. A better option in that scenario is to use a credentials file, which is simply a text file containing a user name and password.

However, that method is *not* appropriate for storing any credentials you actually use to log in to systems. Even with strict file permissions set, it may be possible for some unauthorized person or process to copy or read the credentials file.

As with any type of filesystem mounting, you can save yourself typing by creating an entry for your mount in `/etc/fstab`. For the example we just used, a corresponding `fstab` entry would look like this:

```
//CASA_DE_MICK/BUZZ-OFF /home/mick/mymountpoint cifs  
rw,noauto,suid,user=mick 0 0
```

As you can see, this line is very similar to the `mount` command line we used earlier. One new option here is `noauto`, which causes this line to be ignored at system startup—this Samba share won't be mounted until you issue a `mount` command, like this:

```
myclientlaptop-$ sudo mount /home/mick/mymountpoint
```

`sudo` will prompt you for the root password. (Again, if you aren't running Ubuntu, you could omit the `sudo` command and instead execute the rest of the command from a root shell session.) Then, `mount` will prompt you for mick's password.

Assuming authentication succeeds, you'll be able to use `BUZZ-OFF` as if it were part of your local filesystem, located in `/home/mick/mymountpoint`. When you're done working, you can unmount it like this:

```
bash-$ sudo umount /home/mick/mymountpoint
```

If you prefer your Samba mount to be activated every time your system starts, you can omit the `noauto` option in your `fstab` entry. However, unless you use a credentials file, you'll need to be present during each startup in order to enter the Samba password when prompted; otherwise, your startup will wait for you indefinitely. On a laptop system this probably isn't a problem, but on other types of systems it very well could be an issue.

Similarly, if your Samba server is unavailable for some reason when your client system starts up, this also can cause the client startup to hang or delay. When in doubt, stick to `noauto` mounting.

Samba Security

By Mick Bauer

(Reprinted From the Linux Journal)

Conclusion

And, that's it for this series on Samba security. Funny how four columns can add up to only a basic tutorial, but I hope you've found it useful. Until next time, be safe!

Resources

"Samba Security, Part I", *LJ*, November 2008: www.linuxjournal.com/article/10224

"Samba Security, Part II", *LJ*, December 2008: www.linuxjournal.com/article/10256

"Samba Security, Part III", *LJ*, January 2009: www.linuxjournal.com/article/10292