How I Use Microsoft Access User-Level Security

This document describes my methods for using Access database security. You might ask: "Isn't there already enough information written about Access security? Why write another paper?" Well, from the continuous stream of Usenet queries about security, it's clear that many people don't understand how to make it work. The Microsoft FAQ (http://support.microsoft.com/support/access/content/secfaq.asp), which was written by some of the most knowledgeable Access authors around, is the definitive source of information, but it has some problems in my opinion. For example, it deals with over fifty questions, many of which are quite technical, but only about 10% of the document covers the basics. My goal is to expand the coverage of the basics. It also recommends that you use the Security Wizard, which reportedly leaves some security holes if used incorrectly. I don't use the security Wizard, so this paper will describe strictly manual methods.

When Access 97 was released, many articles about the Access security model were published in trade magazines. That's old hat now, so the Access security model doesn't receive the same amount of publication space in the magazines as it once did. The magazines would rather cover the newer technologies, so beginner Access developers are left without that information source. I hope this document will fill some of that information gap.

Also, I've learned a lot from the online community, and this is my effort to repay that knowledge.

Before starting, let me say that I've worked with Access 1.0, 1,1, 2.0, 95, and 97, but only recently started using 2002, and have never even *seen* Access 2003. There are some variations with implementation for these latest versions, although the concepts are identical. I mainly discuss *what* to do, not *how* to do it, so you may have to be flexible, and adapt to slight variations in implementation.

The three pillars of security

I believe that understanding the basic workings of a computer technology will make it easier for you to use. This is in contrast with the Microsoft/Windows approach, which presumes you can't understand anything technical, so all the details are obscured within wizards. I won't coddle you – I expect that you will expend some effort to understand what's happening behind the scenes so you can better understand how to use the software. If you don't like this approach, stop reading right now, and go back to using the wizards.

I will start with a review of Access's security model. The Access security model, or more correctly, the Jet security model, uses three components to keep your data secure.

The Jet database engine

Right about now you might be asking "What the heck is Jet? My database manager is Access!" Well, it turns out that you are not really using an Access database after all because Access is not a database manager! Instead, Access is a programming environment and user interface that can *connect to* database engines. The confusion arises because, as a retail product, Access is packaged with Microsoft's database manager called Jet. Microsoft makes no attempt to make you aware of the difference. Sometimes it's hard to distinguish one from the other, but make no mistake, they are two separate and distinct programs. Jet manages the data and Access lets you, well, access it. In truth, you can use Access without Jet, and you can use Jet without Access. They are different animals.

Why belabour this point? It's important because whenever you access Jet data using the Jet database engine, Jet's security is involved. You can't get around it, whether you use the data from Access or from Visual Basic or from Excel or from any number of other programs that can use Jet. If Jet is involved, then Jet's security will be invoked.

Another reason this is important is that if the Jet database engine is *not* involved, then its security will *not* be invoked. How might this happen? Suppose some hacker copies your MDB file onto his computer, and then uses some low-level tools to scan your file, thus bypassing the Jet engine. All your data would be exposed.

The bottom line is that the Jet database security system can prevent casual users from making blunders and "knowledgeable" users from snooping. However, it is not strong enough to keep out determined hackers. If this raises security concerns because of your data's sensitivity, then it's time to look at a different product. Jet is great as far as it goes, but its fundamental design as a desktop database manager keeps it from achieving the security levels of industrial-strength database managers such as SQL Server or Oracle.

The workgroup file

The workgroup file, (default name is System.mdw, probably in your Windows\system folder or your user-specific Windows setup folder) lists all of the user names, group names, and passwords on your system. It's the component that tells the Jet engine who you are, and is used when Jet decides whether or not to allow you to perform each database function. Every Jet session uses exactly one workgroup file, and you typically log onto the workgroup file when you start Access. Once you start Access, your identity is established in the system until you quit from Access. I'll talk about joining different workgroup files a little later.

The important thing to understand is that the workgroup file just identifies *who you are*, but it does not determine what you are permitted to do. For that, you need the third component of the Jet security system.

The database

Every object in every MDB file is associated with a set of permissions. Every table, every query, every form. Everything. Even the MDB file itself has a set of permissions. There is nothing you can do in a Jet database without invoking the security system. Every action and object is always checked. You simply can't get away from it!

When you try to use one of these objects, say, to read a table, Jet consults the list of permissions for the object and compares them to who it thinks you are (as determined by the workgroup file). If you have sufficient permissions, then Jet will complete the action. If you do not have sufficient permission, then you are denied access to that object.

Security on my computer appears to be turned off

You may well be saying: "But anybody can use the Access database that I just created, so the security must be turned off on my copy of Access." Well, it's true that anybody can use your database, but it's just an illusion that security is not involved. What's really happening is that the object permissions in your database are so lax by default that it *appears* that security is turned off. However, there simply is no switch in Jet to turn security on or off – the only way to affect security is to modify the permissions on the database objects.

Microsoft made some deliberate and powerful choices when they designed the Jet security model. These choices allow the casual user to work as if the security system was turned off, yet allow the sophisticated user full access to some sophisticated security features. And both these scenarios have the security system

operating in full force! Clever people, those Microsoft programmers! However, this duality of security can lead to traps for the unwary, so you must be careful to use security properly.

The important point to remember is that permissions are stored with the MDB file. In fact, you can use any number of workgroup files to access an MDB file because the workgroup file does not determine your permissions on the database objects. This point can come back to haunt you if permissions are configured incorrectly in the MDB file.

To reiterate: Whether or not Jet will allow you to access the objects in the MDB file depends on your userID in the session's workgroup file and the permissions stored in the MDB file itself. See how they all work together?

It's not a database password

There is another completely separate security system built into Jet: the database password. This system allows you to assign a single password that must be supplied before the database can be opened. However, it doesn't offer any method to distinguish between users, and it is (apparently) quite easy to hack. Most Access developers will agree it is next to useless. Don't even bother thinking about it.

To reiterate: Jet security is *not* a database password. If you are using a database password, you are wasting your time and this document is not for you. Enough said.

Use a single workgroup file

So now you understand the three pillars of Jet security. What's next? Well, you can't do much about the Jet engine – it's either installed on your computer or not, so there's no more to say. Let's start with the workgroup file.

Installing Access on your computer will create a workgroup file called System.mdw, probably in your Windows System folder or your user-specific Windows setup folder. It's the default workgroup, and it's the only workgroup that you should ever join. You should *make* other workgroups, but you shouldn't *join* them. I'll clarify that shortly…

You can either make one workgroup file for *all* of your secured applications, or individual workgroup files for each application. Both strategies have merit, but to start with, just make one workgroup file for all your applications, then modify that strategy after you've outgrown it. You will know when that time arrives.

Before making the workgroup file, think about the people who will use your secured databases. They *all* need to be able to see the workgroup file, so select a shared network location where *everybody* has read-write permissions (I'll talk about network permissions later). Bottom line – save yourself maintenance headaches by putting the secured workgroup file in the right place to begin with.

Also, name it appropriately, say, ShareWorkgroup.mdw. Whatever you do, don't reuse the System.mdw name because that will only cause confusion. Also, avoid names that conflict with your database names. Access won't run properly if the workgroup name (MDW file) and the database name (MDB file) are identical.

Protect the PID

OK, you've decided on a workgroup filename and an appropriate network location. You need to run the workgroup administrator program. It's a standalone

program under Access 97 (Wrkgadm.exe, probably in your Windows folder), but it's accessed via the Security menu in Access 2002. I'll just call it WorkGroupAdminProcess for consistency here. WorkGroupAdminProcess will ask for three pieces of information to create the workgroup file – WRITE THEM DOWN SOMEWHERE SECURE. If you ever need to recreate the workgroup, you can do so *only* if you have this information. WorkGroupAdminProcess advises you to save this information. It's important. Write'em down *before* you create the workgroup.

Join a workgroup the hard way

Whenever you open a Jet session, you are "joined" to exactly one workgroup file. The Windows registry tracks the location of your default workgroup and (usually) uses that file whenever you start Access.

By default, WorkGroupAdminProcess will change your Windows registry setting so you are automatically joined to any the new workgroup file that you create. I recommend that you re-join the default, unsecured workgroup file (system.mdw, probably in your Windows\system folder or you user-specific windows folder). If you do that, then you know that whenever you run Access using its normal shortcut, you will be using it "unsecured". You need to make a new shortcut to run it "secured".

Use a shortcut to use the correct workgroup the easy way

Here's something that Microsoft won't tell you: You should *always* be joined to the default, unsecured workgroup. Whenever you start Access using the default shortcut, you should be using the same "unsecured" configuration as your applications's users.

Joining workgroups manually using WorkGroupAdminProcess is a waste of time and effort. Instead, you should temporarily join a secured workgroup *for a single session* by using appropriate shortcuts that use command-line parameters. Essentially, you encode the database name, the workgroup filename, the user, and the password into the shortcut so that you never have to enter that information manually. I store my shortcuts in a secure location (i.e., my home directory on my secure network), so I don't have to worry about prying eyes.

The trick with these shortcuts is that you must also include the full path name of the Access executable file – it's not sufficient to just enter the name of the database file like you might do with, say, a Word or Excel document. The first item in the shortcut is the Access executable, the second item is the database name, then followed by the optional items. *If you omit the path to the Access executable file*, then you will always be joined to the default workgroup as determined by your Windows registry. You must include the path to the Access executable in the shortcut. See the Help file for details about the syntax.

This is the *only* practical way to get your end-users to join the workgroup file. Don't even think about asking them to join the workgroup manually because it's way too much work.

I use several shortcuts – some join the workgroup and launch specific applications, some just join the workgroup as a particular user, some just join the workgroup and ask for a login ID. One of the most useful shortcut simply joins the secure workgroup and starts Access. Create that one first since you're going to need it shortly.

Make life easier for yourself and your users – create shortcuts to join the workgroup temporarily. Here are the "target" properties of several shortcuts I use

all the time (watch out for word-wrapping, spaces between parameters, and spaces in folder names):

Your first shortcut should resemble this one:

```
"C:\Program Files\Microsoft Office\Office\MSACCESS.EXE" /wrkgrp
"d:\jack\@Coding\macdonaldsoftware.mdw"
```

This one logs me on as a specific userID:

```
"C:\Program Files\Microsoft Office\Office\MSACCESS.EXE" /user Jack
MacDonald /wrkgrp "d:\jack\@Coding\macdonaldsoftware.mdw"
```

This one logs me on without asking for a password:

```
"C:\Program Files\Microsoft Office\Office\MSACCESS.EXE" /user Jack
MacDonald /wrkgrp "d:\jack\@Coding\macdonaldsoftware.mdw" /pwd
"xxxxx"
```

This one logs me on without asking for a password (the LODMadmin user's password is blank) and it opens the MDB file. The MDB filepath must be the second parameter:

```
"C:\Program Files\Microsoft Office\Office\MSACCESS.EXE"
D:\Jack\WORK\ValleyTransport\LubeOilDistributionManager.mdb /user
LODMadmin /wrkgrp "d:\jack\@coding\macdonaldsoftware.mdw"
```

This one uses a different workgroup file:

```
"C:\Program Files\Microsoft Office\Office\MSACCESS.EXE" /user
FERICProgrammer /wrkgrp d:\jack\work\feric\address\FERICwe.mdw /pwd
"yyyyyy"
```

Notice that all the shortcuts actually point to the location of the Access executable file. Each reference to a file must conform to Windows file-naming standards – watch out for spaces in names. It may be worthwhile to enclose the filename in quotation marks. *Always* use quotations if the path name includes spaces.

Consult your Windows documentation if you need help creating shortcuts. Consult the Access help for additional information about command-line syntax.

The Admin user – who *is* this person

OK, you've made a new workgroup file and a shortcut to join it temporarily. But nothing seems different when you launch the shortcut. Access starts as normal. What's going on?

This is where the lax security that I mentioned earlier comes into play. You logged onto the new workgroup as the "Admin" user. Admin hails back to the first versions of Access, and my guess is that somebody at Microsoft feels guilty about that name (that is, if anybody at Microsoft *ever* feels guilty about anything…). A better name would have been "Anybody" or "Everybody" or "This is a gigantic hole in security". Anything but "Admin" which connotes some special status. Yechh. Don't believe it – Admin is not special, that is, Admin is not special once you've plugged the existing security holes.

Admin is the default user. You cannot delete the Admin user from your workgroup file. Admin is also a huge security hole that you must render powerless. Limiting Admin's permissions on your database is one of the most important things you must do to set up security properly.

What causes Access to ask for a username and password?

Unless you change something on your system, you will never be asked to supply a name and password, and you will always silently log onto the workgroup file as the Admin user. The "something" is easy – just implement a password for the Admin user. Once that's done, then Access will ask for a userID and password. It's that simple.

To reiterate, until and unless you supply a password for the Admin user, Access will always silently log onto the workgroup file as the Admin user. So do it now – assign a password to the Admin user on your secure workgroup file. Oh yeah, write it down – you might need it later.

By the way, if you haven't figured it out already, all the security routines are accessed via the Tools > Security menu. Ignore any wizards, and just poke around in the menu system for a while to get a feel for what commands are available.

Every Admin user is identical

The problem is that *every* Admin user from *every* workgroup file from *every* copy of Access in the world is identical. There is *nothing* you can do to make your Admin different from my Admin. That's the way Microsoft made it, and that's the way it will stay. Get used to it.

So if you leave a security hole that *your* Admin can exploit, then *my* Admin can exploit the same hole. Or more to the point – any of your users who silently log onto their default workgroup file as Admin can also exploit the hole. That's just the way it is, but we're going to close that hole shortly.

Almost every group is different

So, now you know that permissions granted to the Admin user are portable across every workgroup. The same cannot generally be said about Groups (not quite true, but close enough for now). If you define a group in two different workgroup files, then they will usually be recognized differently by your database, and their permissions will be different.

The Users group is an exception. Every Users group is recognized identically by every Jet database. So, if you leave a security hole that your Users group can exploit, then your silently-logged-on Admin users can exploit the same hole because Admin cannot be deleted from the Users group. As with the Admin user, the Users group cannot be deleted. It will always be in yours and every unsecured workgroup file's User group. Therefore, it's up to you to manage the database permissions properly for the Users group.

The Admins group is another exception. Members of the Admins group can manage the list of user accounts in the current workgroup, thus, they may be able to put themselves into a group that has more database permissions than you really want them to have. In other words, they can bypass your security settings. Fortunately, the Admins group is not portable across workgroups, but it's a security hole that needs to be plugged nonetheless.

Now you know about the second major hole to be plugged in the Jet security system. We will get to the "how-to" shortly.

How can users from different workgroups be identical

What's going on? Why are Admin and Users identical in different workgroups?

Well, to understand the exceptions for identifying users, you must understand the general rule. When you log onto a workgroup file, Jet uses the workgroupID, user name, and PID to create a unique identifier called the SID ("security ID"), which Jet uses to verify your permissions in the database file. The workgroupID is the "secret" word that you used to create the workgroup file. The PID is the Personal ID that you used to create new userID and groups.

So if you create two workgroup files using the same identifiers, and create userIDs within each workgroup file using the same names and PIDs, then they both generate the same SID and Jet cannot distinguish between the two userIDs. For all intents and purposes, they are the same because they have the same SID.

Turns out that the Jet database engine generates the same SID for the Admin user and the Users group regardless of the workgroup file that Jet is currently using. There is simply no way for your database to distinguish between the Admin user and the Users group emanating from any workgroup file in the world. They are all the same. Microsoft had good reasons for making it so, but it can be very confusing unless you understand what's going on!

Predetermine the users' access requirements

We are getting closer now. The key to managing Jet security is to properly manage the userIDs, the group names, and the database permissions. Let's start with group names.

Create and use consistent group names

As a general rule for managing any computer security system, you should grant permissions to groups, not to individuals. Same principle applies for Jet. You need to set up groups for which you can manage permissions. You can subsequently add or delete users from the groups with a minimum of fuss.

You should develop a strategy for creating the group names. I use three groups for each application (or group of applications). Depending on circumstances, you may require more groups for each application. Of course, when two applications have the same body of users, they can share the same user groups. Start with three groups, and work up from there.

Be sure to write down the PID whenever you create a user or group. You will need that information if you ever want to recreate the user or group.

Users

These are the everyday users of your application – the people that need to create, read, and edit the data. Create a users group for each application, e.g., SalesUsers, AccountingUsers, EngineeringUsers, etc. I will refer to these groups generically as "applicationUsers".

Administrators

These are people who need to modify "configuration" files of your application (more about that later). Stuff you don't want every Tom, Dick, or Sally mucking with. Create an administrator group for each application, e.g., SalesAdmins, AccountingAdmins, EngineeringAdmins, etc. I will refer to these groups generically as "applicationAdmins".

Developer

These are your "super users" – the ones with the keys to the vault. These are people that need permission to modify the database structure. Your database probably also contains "system" tables that nobody but the developers should modify. You're reading this, so you are probably a good candidate for being in the Developer group. You might consider creating an "applicationDeveloper" group, or just a "Developer" group. It's up to you. I will just refer to the "Developer" group.

Create the super user account

Groups are useless until you create some userIDs. For now, you will just create just one userID – the super user of your system. You will create more accounts later, but just one will suffice for now. You are the super user – the one that controls everything in the database. Enjoy the power!

Create that super user now. If you are still lost trying to find the appropriate commands, look under the Tools > Security menu. As I said before, I will concentrate on *what* to do, not how to find the appropriate menu commands. Poke around in the menu system for a while, and you will find everything that you need. Experiment!

Notice that Access automatically adds the super user into the Users group.

Super user joins the Admins groups

Besides the Users group, Jet also creates the Admins group that cannot be deleted. Members of the Admins group can manage the user accounts in the current workgroup. Add the super user account to the Admins group. While you're at it, add the super user to the Developer group. If you feel like creating some additional work, add super user to the applicationUsers and applicationAdmins groups, too. Not strictly necessary, but no harm done, either.

Say goodbye to Admin and become the super user

Quit, and then restart Access. Notice that it asks for a userID and password – and logon as your super user. The UserID won't have a password yet, so that's the first thing to do once Access has started – create a password for the super user. You could save some future keystrokes by making a new shortcut that joins the secure workgroup under the super user's name. If your shortcut is secure, consider adding the password to the shortcut so you can log into your secure workgroup quickly and easily (see the previous examples).

Remember Admin? It's still a member of the Admins group, and therefore represents a huge security hole because Admin could simply add itself to another group. Remove Admin from the Admins group – you couldn't do that until you were logged in as super user. Double-check that Admin is a member of *only* the Users group. Consider the Admins group to be used *only* for managing the list of user accounts. User accounts belong in the Admins group *only* if you want them to be able to add or delete user accounts from your various groups.

This point is worth repeating: Admin should be a member of ONLY the Users group.

Do it manually – don't trust the wizard

I must confess – I have never used the security wizard. It may be the best thing since sliced bread, but Usenet carries *lots* of cries for help from people who have used the wizard and can't get anything to work properly. Methinks something is rotten in Denmark… The wizard probably does the following steps automatically, but for me, I'll just continue to do it manually. My advice is for you to do the same.

I look at it this way: the manual method probably takes longer to learn and implement than using the wizard – *if* everything works right with the wizard. But if you waste three days fooling around with the wizard because you missed a step and don't really understand what's going on, then what have you saved? Nada… You would have been better to do it manually the first time. I simply don't believe that Jet security should be used by "developers" who don't understand how it works – and that's the foundation of the security wizard.

The database owner

So… we've come to the point where we've created groups and the super user account, and have assigned the super userID to the Admins and Users groups. Next step is to deal with the third pillar of Jet security, and secure the database itself.

The first step to secure the database is to understand the database owner. The database owner is the userID that was in effect when the database was created – likely the Admin user. The database owner can always create new objects in the database and can almost always grant himself any permission to all database objects (there are some conditions beyond the scope of this paper where an Admin database owner can be denied object permissions). In general, if you fail to manage the database ownership properly by removing Admin as the database owner, you will leave a huge security hole. You simply *must* change the database so that Admin is no longer the owner.

Got that? Admin *cannot* be the owner of the database if you want it to be secure!!

You affect the database owner by logging onto the workgroup file under the userID of the prospective owner and creating a new database. That's why you had to log onto Access as the super user.

Create a new database now and import all the objects from the original database. The reason you are allowed to import the objects from the original database is that you are logged in as super user, and super user is a member of Users group, and the Users group has permission to import the objects. See how big a security hole the Users group is?

You've now got a database for which the super user is the owner and which contains all your database objects.

The object owner

Each database object also has an owner, but unlike the database itself, you are allowed to change their ownership. Admin is likely the owner of all the database objects, and you must change that or you will leave another security hole open to the world. Change the owner of every database object to your super user. As you are poking through the "change owner" part of the menu system, notice that you cannot change the owner of the database. Can't be done. That's why we had to make a new database in the previous step.

Database object permissions

OK – you've now created the user accounts and groups, and you've set up the database and object owners. Next step is to set the permissions on individual objects.

Grant permissions to Groups

Time to plug one more security hole: the Users group. To lock out the Admin user, you must remove all permissions from the Users group. However, before nuking those permissions completely, you should think about whether you want the Admin user to have *any* permission at all. Remember, anybody silently logged onto Access using their unsecured workgroup is the Admin user – it's up to you to decide whether they should have any database permissions.

For example, I sometimes grant Read Data permission to the Users group so that anybody running any copy of Access can see the data. That philosophy may or may not be appropriate for your situation. Regardless of the choice you make, it's via the permissions on the Users group that you invoke your decision. If you want the Admin user to have some access to the data, then grant the User group some permissions to the tables. If you don't want "lurkers" in your database, then lock 'em out. It's up to you…

Next, you need to set the permissions for your primary groups, namely the "applicationUsers", "applicationAdmins", and "Developer" groups. This can be tedious, and a trick that I've learned is to think about this step *long before* building the database. For every table in your database, you should know who needs to edit the data – *that* should be part of your database-design process. I routinely name the tables according to the designated "user" of the table. For example, if the table should be editable by any authenticated user, then the table name begins with "u". If it's an administrator table, then the table name begins with "a". Finally, the developer's tables begin with "d".

Why bother with this scheme? It turns out that the Access window where you assign the permissions always sorts the objects alphabetically. Using this consistent naming convention causes all the similar tables to group together. It's very easy to identify and grant the correct permissions using this system. Otherwise, all the tables will be jumbled together, and it will be a mess to assign the permissions.

(An aside: If you've already built your database and think it would be a hassle to track down all the references to your table names throughout your application, here's an alternative. Rename all the files in the backend according to my scheme. Link the frontend to these renamed tables, but retain the original name in the frontend. Your application won't know the difference. Alternatively, rename the tables in both the frontend and the backend according to my scheme, and then create some Select queries using the original table names that point to the newly-created tables. But for your next database, put some thought into using this scheme before you create the tables.)

You should also consider what permission to assign to the "new table" object. I usually make it an "applicationAdmins"-type table, so that any new table defaults to the "applicationAdmins" permissions. But there's no correct answer – you will always end up checking and changing the permissions for some new tables.

Be sure to grant all permissions to all objects to the "Developer" group. That will be the route by which your developers will be able to modify the database objects. It's a bit redundant because super user is the database owner, and thus has permissions on all objects, but I feel better about tidying-up the loose end.

Besides, this gives you an avenue whereby you can add other userIDs to the "Developer" group and implicitly grant them permission to modify the database objects.

Finally lock out the Admin user

Remove all the permissions for all users on all database objects. Granting permissions to individual users is generally a bad idea – you grant permissions to groups then assign the users to groups. At this point, you should only have two userIDs – Admin and your super user. Neither of them requires any explicit database permissions.

In case you missed the point – you finally locked out the Admin user. Goodbye Admin, and good riddance!

Least restrictive permission wins

In the Jet security model, the least restrictive permission wins. For example, a logged-in user with read-write permissions on a table granted from one source (say, via membership in the "applicationUsers" group) and read-only permission on the same table from a different source (say, by belonging to the Users group) will have read-write permissions on that table. Therefore, you must carefully consider all group memberships when a user belongs to more than one group.

Create the remaining user accounts

Hey – it's finally time to talk about the individual users of your database!

This is another place for you to apply discretion: you could create a separate userID for every person who needs to access the application. That strategy provides the maximum security level, but also brings with it certain maintenance requirements. Alternatively, you could create fewer user accounts, and require the users to share the passwords. You could even encode the user accounts and passwords into shortcuts, and distribute the shortcuts to the appropriate users. This approach provides less security but also requires less maintenance. You will have to evaluate your own situation.

Create user accounts for your everyday users and your application administrators.

Assign users to accounts

Assign all the everyday user accounts to the "applicationUsers" group, the administrators to the "applicationAdmins" group, and the superusers to the "Developers" account.

You will notice that Access added every userID to the Users group. That's OK – because we already plugged the Users group security hole.

Notice that you don't have to deal with granting permissions to individual users. You've done it once for the groups, and you never have to concern yourself with it again! Hooray!

Deploy the database

Congratulations. You've done it. Your database is now secure. You've created a workgroup file and database file in the appropriate network locations, and you've created an appropriate set of user and group accounts. You've created shortcuts that join the workgroup file for the duration of a single session and launch the

database. Finally, you've assigned ownership and permissions to the database objects to let the good guys in while keeping the bad guys out.

Next step is to deploy the shortcut files and allow the users to use the database. Good luck.

Other issues

Use network security as appropriate

All this security stuff makes most sense on a network where multiple users are involved; you should take advantage of the network's own security system. If you mount your MDB file on a network drive that only limited people have access to, then you've just increased your security level beyond what Jet provides itself.

But remember that Jet requires full read-write permission on the network share where you mount the MDB file. If the network drive does not provide the user with read-write access, then your application just isn't going to work.

The workgroup file also must be in a network location where every user has full read-write permissions.

What to secure – front end or back end

All serious Access developers recommend that you split the database into two components – a backend that contains all the tables and a frontend that contains everything else. Splitting the database provides some significant advantages, including performance, ease of updating the application, replication, and stability.

But which of these databases should you secure, and how? I look at it this way: security on the backend is about protecting the integrity of the *data*. You must get it right, because the data is the most valuable part of your application. On the other hand, security on the frontend is about protecting the integrity of your *application*. If a user corrupts the frontend, it can simply be replaced, especially if you follow the conventional advice and install an individual copy of the frontend on every user's desktop. It may be a hassle if the frontend becomes corrupted, but it's not mission-critical.

Let's discuss the backend security first.

A basic principle is that tighter security on the backend will override looser security on the front end (this is opposite the principle that the looser permission prevails, but that's the way it works!). In other words, if a user has read-only permission for a table in the backend, then granting read-write permission on the frontend will *not* compromise the integrity of the data. The user will continue to have read-only access on the table. On the other hand, if a user has read-write permission on the backend, but you grant read-only permission on the frontend, then that user will have read-only permission, which is not what you want. Tracking down that bug would be difficult…

Therefore, apply the correct table permissions on the backend according to the user's requirements.

Things are a little different with the frontend because there are two types of objects to consider: linked tables and everything else. Linked tables in the frontend are simply symbolic connection strings to tables in the backend. Granting full read-write permissions does not affect the data integrity, but it *does* allow the user to re-establish those links. It's the recommended starting point for linked-tables permissions in the frontend. This stuff gets quite technical, and is

well-explained in the Microsoft security FAQ. If you're dealing with these issues, then you've advanced beyond the scope of this paper. Go read the FAQ.

Bottom line: grant all your users full permissions to the linked tables in your frontend database unless you have a specific reason not to, as outlined in the FAQ. Doing so will not comprise the security of your data in the backend.

For all the other database objects, you will have to evaluate them according to your own needs. You can also create an MDE file from the frontend to provide additional security for your intellectual property. One point: granting full read-write permissions to all queries does *not* affect the integrity of the data in the backend. I find it easiest just to grant full permissions to all queries, and allow the backend to control the permissions.

(Here's one tidbit from the FAQ that took me a long time to understand. It's a bit technical, so you may want to skip it for now. Users require ReadData permission on the backend to relink the frontend and backend using the RefreshLink method. If they don't have ReadData permission, then they must use the TransferDatabase method. The Access Linked Table Manager uses the RefreshLink method, so is affected by this detail. You would only be concerned about this if you need to deny some of your users permission to read some data on the backend.)

Database passwords

As I said earlier, don't bother with database passwords. They simply get in the way and are a hassle to manage. The user-level security is much better.

If your goal is the simplicity of a single password, you can simulate the effect of a database password by creating a single userID and providing your users with a shortcut that logs onto your secured workgroup under that single userID.

An aside: I ran into a situation where the developer applied a password to the frontend, but no security to the backend. The objective was to keep out casual snoopers. All the snooper had to do was to open the backend!! The password offered no security at all. Don't fall into the same trap.

Different row/column permissions for different people

Permissions are applied at the table level. In other words, you cannot directly configure Jet to allow some users to read or write to the whole table while restricting other users to just selected rows or columns. You have two choices if you need to apply that level of granularity: you must either abandon Jet for a different database manager or investigate so-called Read With Owners Permissions (RWOP) queries. RWOP queries provide you with a means to achieve row-level or column-level security with Jet, but require a level of sophistication to your application that is beyond the scope of this document. Read the FAQ for additional information.

When Jet is not enough

As mentioned near the beginning of this paper, Jet is a desktop database manager that does a good job of the tasks for which it was designed. However, it *is* possible to crack Jet's security using tools that are available inexpensively on the Internet. Database encryption can even be broken easily, if you were thinking about applying encryption. If air-tight security is your requirement, then you should look at alternatives to Jet. But if keeping out the casual snoopers is your goal, then following the steps laid out here will see you through.

Some Common (or not!) Problems

The correct way to fix these problems is to secure your database using all the steps outlined in this paper, in the order they are presented in the paper. Unfortunately, there are no shortcuts, but these review points can help you know where to start.

My database is secure on my computer but wide open on other computers on the network

Your network users are silently logging onto Access as the Admin user, and you have failed to limit the Admin user's permissions on the database. There are several ways this could have happened:

- The Admin user continues to be the database owner, and the owner cannot be denied access to its own database. You must make a new database for which Admin is not the database owner.
- The Admin user continues to be part of the Admins user group. Admin user must be removed from the Admins group. As a practice, I also deny permissions to the Admins group because I prefer to use the Admins group *only* for managing the list of user accounts in the current workgroup file.
- The Users group has excessive permissions on the database objects. Admin can never be deleted from the Users group. Depending on your objectives, you can either deny all permissions to the Users group, or allow just certain, restricted permissions to the Users group.
- The Admin user has excessive permissions on the database objects. As a matter of good practice, you should seldom (if ever!) grant permissions to individual users. Instead, you should manage permissions with groups, and then assign the users into groups as appropriate.
- The Admin user is a member of another group that has excessive permissions. Never assign the Admin user to any group other than Users.

I've secured my database, but now Access asks for a password to open every database

It's the *workgroup file*, not the database file, that controls whether or not Access asks for a password. You have joined a secured workgroup file. The solution is to rejoin the default workgroup file (which should have no user accounts besides the Admin user), and to use the secured workgroup file on a per-session basis using a customized Windows shortcut file.

My network users don't have any permissions

Unlike the Windows shortcuts you create for, say, Word or Excel files, the shortcuts you create to launch an Access database *must* include the path to the Access executable file. If you omit the Access executable file, then the Access session will use the default workgroup file, and your users will probably be identified as the Admin user. The solution is to properly construct a Windows shortcut that includes the Access executable, the MDB filename, and the workgroup filename. You can optionally include the userID and password in the shortcut.

Unsecuring a database

Unsecuring a database requires that the Admin user has full access to all the database objects. Remember, every Admin user on every workgroup file appears identical to the Jet database engine, so as long as the Admin user has permission to use an object, the database will appear to be "unsecured". Log in using your superuser account and grant full permissions to the Users group and the Admin user. Put the Admin user back into the Admins group and then remove the password from the Admin user. You may even consider making a new copy of the database with Admin as the owner (the reverse of the step required to eliminate Admin as the database owner in the first place.)

Replication Manager and User-level Security

If you completely lock the Admin user out of your backend database, then you will be unable to open the database using Replication Manager unless you use a command line switch. Replication Manager supports the same command-line syntax for workgroups and users as does Access itself.

What workgroup file is active

Here's a quick way to determine what workgroup file your Access session is joined to: Open Access as normal, then press Ctrl-G to open the VBA Immediate window. Type "?dbengine.systemdb" (without quotes) and press Enter. You won't be able to do this if you application blocks your access to the Immediate window.

Local administration of user accounts

If you distribute your application to many sites, you won't want to manage the user accounts yourself – somebody at the site should do that task. But how do you provide local autonomy for creating accounts, yet deny them the ability to muck with your databases? The key is to create two workgroup files – one for yourself as developer, and one for distribution to the sites. Create the same groups in both files except for your superUser account and Developers group – keep those to yourself. Use the same names and PIDs when creating the groups so that your applications see them as identical. Deny permissions to the Admins group to all your database objects – this will allow the Admins group to administer the user accounts, but not to manipulate the database objects. Create a standard user name (e.g., AccountManager) to distribute with the workgroup.

The local administrator will be able to log onto the workgroup file and create all the local user accounts and assign them to your groups. But they won't be able to modify the permissions for the groups. They will be able to create new groups, but it won't do them any good because they won't be able to assign those groups any permissions in your databases.

Of course, you will send them the workgroup file only *once*. If you sent them a new copy, it would destroy all their user accounts (including passwords), and you would not be very popular with the remote users. Therefore, it's vitally important that you set up the necessary groups *before* you distribute the workgroup file to your remote sites.

Only one person can open my database at a time

Access automatically creates and deletes a "lock" file in the same folder as the MDW and MDB files as users log onto the workgroup and use the database. If

Access can't manage the lock file (.LDB) properly, then you will experience problems.

All users must have full read-write-create-delete permissions on the network share where your database and workgroup files are stored. To test whether the users have sufficient permissions, try creating, modifying, and deleting a text file in the appropriate network folder using Notepad while logged onto the network as the appropriate user. Any difficulties using Notepad will cause Access to fail to work with a file in that folder. If you don't manage the network yourself, ask your network admin for help, and emphasize that the users need *full* permissions.

You can safely delete the .LDB file once everybody has exited from the database. Deleting the .LDB file sometimes clears up strange "locking" behaviours.

Conclusions

I hope this paper clarifies some issues around Access User-Level Security and makes it easier for you to secure your databases properly. There is lots of help online in the microsoft.public.access.security Usenet newsgroup.

This paper is available online at http://www.geocities.com/jacksonmacd/

Jack MacDonald

Updates

Original – December 27, 2002

Updates – April, 2003 October, 2003

March 2, 2004

Clarified the location of WorkGroupAdminProcess in later versions of Access.

Cleaned up information about Admins group.

Info about distributed workgroups.