Securing Your Zen/PSQL Database

A White Paper From

GOLDSTAR SOFTWARE

www.GoldstarSoftware.com

For more information, see our web site at
http://www.goldstarsoftware.com

Securing Your Zen/PSQL Database

Last Updated: 02/05/2021

When it comes to data security, the rules are changing quickly. Legislation around data protection in the health care (HIPAA) and criminal justice (CJIS) arenas has presented challenges to some developers for many years now. More recently, GDPR (in Europe), CCPA (in California), and pending legislation in other states will make this an issue for just about every developer creating data-centric applications. It's simply not an option for developers today to ignore security or to try to "bolt it on" afterwards – you must design your environment with security in mind from the ground up.

The Actian Zen database, formerly known as Actian PSQL, Pervasive PSQL, Pervasive.SQL, and even Btrieve before that, offers a number of security options to help you protect your data. While some of these options were included in very early releases (i.e. Btrieve), others were added in Pervasive.SQL V8.5, and still others have been added in the years since.

Let's start by examining a list of the basic security features that are available today:

1. Data File Owner Names
2. Across-the-Wire Encryption
3. SQL Authentication and Access Control
4. Domain-Integrated SQL Security
5. Advanced Btrieve Security

Each security feature protects a different area of the system, so you may find yourself using one or several of these features together. Let's dig a bit deeper and see how each one can protect your data.

Data File Owner Names

The database engine has almost always supported the concept of an **owner name**, or an individual password, on each data file. The owner name allows a developer to restrict access to the underlying file in two different ways:

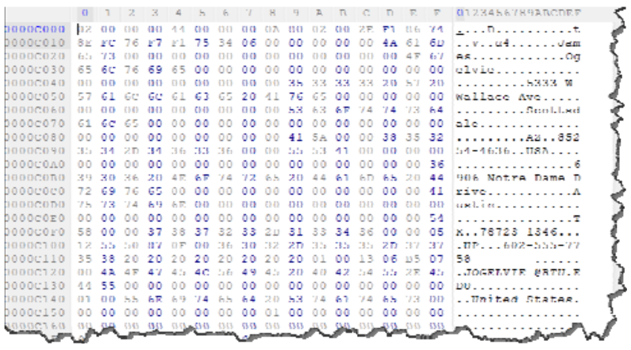
- File Level Encryption
- Access Control

File Level Encryption

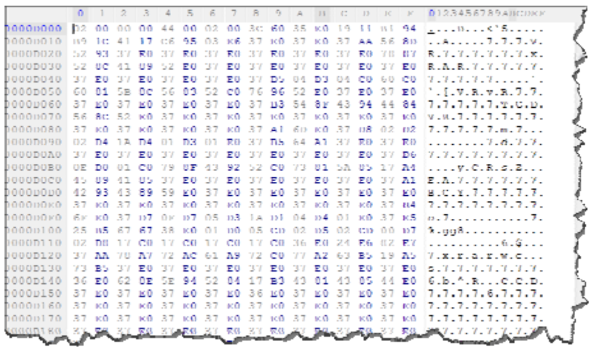
File level encryption leverages an encryption algorithm to encrypt the **data at rest** – i.e. all of the data, index, and variable pages that are written to the disk are stored in an encrypted format. When an owner name with encryption is assigned to a file, the database engine reads, encrypts, and writes back all of these pages in the file, as well as any shadow pages, so that *all* of your data is protected. As the engine reads pages into cache from that point forward, they are decrypted on the fly and left unencrypted in the cache while being used. When pages change and must be written back to the disk, they are encrypted once again on their way to the storage system. Because of the extra steps needed, the encryption/decryption processes will slow down database disk access. However, with a suitably-sized L1 cache, the overhead from the decryption process can at least be minimized.

[*N.b.* Some have asked about the validity of leaving unencrypted data in memory, as this leaves data in the cache subject to attack. This is done for performance reasons, as decrypting EVERY page on EVERY access would be prohibitively slow. Ultimately, data in memory can be attacked by grabbing a core dump of the database engine process, which means compromising the kernel or the server console. As such, physical security of the server itself must always come first.]

Here is an example of a page from a file both without and with encryption, as viewed with a hex editor:



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000C000  02 00 00 00 44 00 00 00 0A 00 02 00 2E F1 06 74  ....D.........t
0000C010  8E FC 76 F7 F1 75 34 06 00 00 00 00 00 4A 61 6D  ..v..u4......Jam
0000C020  65 73 00 00 00 00 00 00 00 00 00 00 00 00 4F 67  es............Og
0000C030  65 6C 76 69 65 00 00 00 00 00 00 00 00 00 00 00  elvie...........
0000C040  00 00 00 00 00 00 00 00 00 35 33 33 33 20 57 20  .........5333 W 
0000C050  57 61 6C 6C 61 63 65 20 41 76 65 00 00 00 00 00  Wallace Ave.....
0000C060  00 00 00 00 00 00 00 00 00 53 63 6F 74 74 73 64  .........Scottsd
0000C070  61 6C 65 00 00 00 00 00 00 00 00 00 00 00 00 00  ale.............
0000C080  00 00 00 00 00 00 00 00 00 41 5A 00 00 38 35 32  .........AZ..852
0000C090  35 34 2D 34 36 33 36 00 00 55 53 41 00 00 00 00  54-4636..USA....
0000C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 36  ...............6
0000C0B0  39 30 36 20 4E 6F 74 72 65 20 44 61 6D 65 20 44  906 Notre Dame D
0000C0C0  72 69 76 65 00 00 00 00 00 00 00 00 00 00 00 41  rive...........A
0000C0D0  75 73 74 69 6E 00 00 00 00 00 00 00 00 00 00 00  ustin...........
0000C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 54  ...............T
0000C0F0  58 00 00 37 38 37 32 33 2D 31 33 34 36 00 00 05  X..78723 1346...
0000C100  12 55 50 87 0F 00 36 30 32 2D 35 35 35 2D 37 37  .UP...602-555-77
0000C110  35 38 20 20 20 20 20 20 20 20 01 00 13 06 D5 07  58        ......
0000C120  00 4A 4F 47 45 4C 56 49 45 20 40 42 54 55 2E 45  .JOGELVIE @BTU.E
0000C130  44 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00  DU..............
0000C140  01 00 55 6E 69 74 65 64 20 53 74 61 74 65 73 00  ..United States.
0000C150  00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
0000C160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000D000  02 00 00 00 44 00 02 00 3C 60 35 E0 19 11 B1 94  ....D...<`5.....
0000D010  B9 1C 41 17 C6 95 03 E6 37 E0 37 E0 37 AA 56 8D  ..A.....7.7.7.V.
0000D020  52 93 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 78 87  R.7.7.7.7.7.7.x.
0000D030  52 0C 41 09 52 E0 37 E0 37 E0 37 E0 37 E0 37 E0  R.A.R.7.7.7.7.7.
0000D040  37 E0 37 E0 37 E0 37 E0 37 D5 04 D3 04 C0 60 C0  7.7.7.7.7.....`.
0000D050  60 01 5B 0C 56 03 52 C0 76 96 52 E0 37 E0 37 E0  `.[.V.R.v.R.7.7.
0000D060  37 E0 37 E0 37 E0 37 E0 37 B3 54 8F 43 94 44 84  7.7.7.7.7.T.C.D.
0000D070  56 8C 52 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0  V.R.7.7.7.7.7.7.
0000D080  37 E0 37 E0 37 E0 37 E0 37 A1 6D E0 37 B8 02 B2  7.7.7.7.7.m.7...
0000D090  02 D4 1A D4 01 D3 01 E0 37 B5 64 A1 37 E0 37 E0  ........7.d.7.7.
0000D0A0  37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 D6  7.7.7.7.7.7.7.7.
0000D0B0  0E D0 01 C0 79 0F 43 92 52 C0 73 01 5A 05 17 A4  ....y.C.R.s.Z...
0000D0C0  45 09 41 05 37 E0 37 E0 37 E0 37 E0 37 E0 37 A1  E.A.7.7.7.7.7.7.
0000D0D0  42 93 43 89 59 E0 37 E0 37 E0 37 E0 37 E0 37 E0  B.C.Y.7.7.7.7.7.
0000D0E0  37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 B4  7.7.7.7.7.7.7.7.
0000D0F0  6F E0 37 D7 0F D7 05 D3 1A D1 04 D4 01 E0 37 E5  o.7...........7.
0000D100  25 B5 67 67 38 E0 01 D0 05 CD 02 D5 02 CD 00 D7  %.gg8...........
0000D110  02 D8 17 C0 17 C0 17 C0 17 C0 36 E0 24 F6 02 E7  ..........6.$...
0000D120  37 AA 78 A7 72 AC 61 A9 72 C0 77 A2 63 B5 19 A5  7.x.r.a.r.w.c...
0000D130  73 B5 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0  s.7.7.7.7.7.7.7.
0000D140  36 E0 62 0E 5E 94 52 04 17 B3 43 01 43 05 44 E0  6.b.^.R...C.C.D.
0000D150  37 E0 37 E0 37 E0 37 E0 36 E0 37 E0 37 E0 37 E0  7.7.7.7.6.7.7.7.
0000D160  37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0  7.7.7.7.7.7.7.7.
0000D170  37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0 37 E0  7.7.7.7.7.7.7.7.
```

As you can see, file level encryption will prevent users from accessing your data by simply reading the data files directly. Because it is an undocumented encryption algorithm, this data is expected to be fairly secure, as well.

WARNING: If you encrypt your data and the file gets damaged in any way, you may not be able to use any of the repair utilities to recover information from the damaged file, because even a single-bit error in an encrypted file can make portions of it unreadable. Having a backup is of paramount importance!

Access Control

The second control capability provided by an owner name is access control. When an owner name is assigned to a file, the developer can either require an owner name for ANY access, essentially locking the user out entirely, or he can allow read-only access *without* the owner name. While this seems silly, the real benefit of allowing read access without an owner name is to protect accounting data or other sensitive information from inadvertent changes, all while allowing users to run reports or SQL/ODBC queries against the data. By allowing read-only access to continue unfettered, the reporting systems can work as expected, and the developer can rest assured that the users won't be changing any data without proper authorization.

Owner Name Levels

These two features are provided by one of four different owner name *levels*:

0. Require an Owner Name for ANY Access, Without Encryption
1. Allow Read-Only Access Without Owner Name, Without Encryption
2. Require an Owner Name for ANY Access, Encrypt Contents
3. Allow Read-Only Access Without Owner Name, Encrypt Contents

Note that these are not in increasing security order, but rather are defined by two bit flags, one used to indicate access control, and one used to indicate encryption. If you look carefully, you'll see that level 2 is actually the most restrictive, since the data on the disk is encrypted and an owner name is required in order to even open the file. Level 1 is the least restrictive, because data is not encrypted and *anyone* can read the data, even without an owner name.

[*N.b.* You may wonder about the value of owner level 3 – why would you encrypt the data, but then allow anyone to read it **without the owner name**? This is more for protecting information from being copied off and accessed with some other tool. Given the pervasiveness of the PSQL engine, though, it takes very little expertise to set up a small engine and access the data directly, so this really provides minimal protection.]

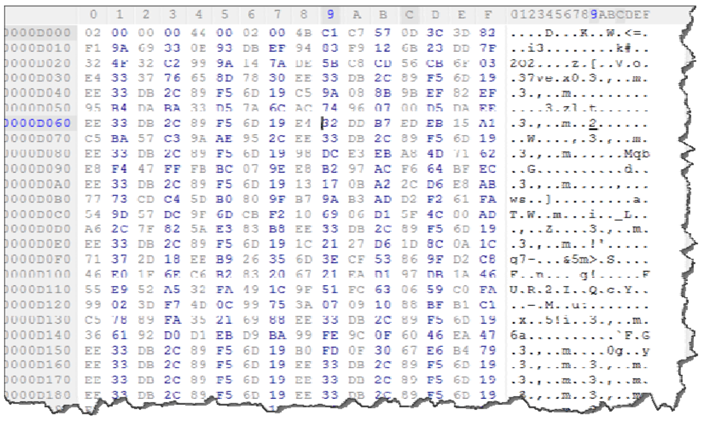
Enabling and Disabling Owner Names

You can enable an owner name on an existing file with the GUI Maintenance Utility using the *Set – Clear Owner…* menu option, using the Btrieve or Btrieve 2 API from an application, or with the BUTIL utility from the command line like this:

```
BUTIL –SETOWNER <filename> /O<owner> <level>
```

Where <*filename*> is the data file name, <*owner*> is the owner name (or *, which allows the user to enter the owner name from the keyboard), and <*level*> is the owner level from the numbered list above. Removing an owner name is done similarly (with the CLROWNER command).

A "standard" owner name is an 8-byte string, usually defined by normal ASCII characters (though this is not a requirement). Pervasive PSQL Summit v10.10 introduced the concept of a *long owner name* – enabled by adding the /L switch to the BUTIL command or an extra checkbox in the GUI – which then allows for a 24-byte owner name string to further improve security. Files with long owner names can ONLY be opened by v10.10 and newer database engines, but this does provide a stronger (128-bit) encryption algorithm, too. Here, you can see the same page shown above, but encrypted with the more advanced algorithm:



```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
)000D000  02 00 00 00 44 00 02 00 4B C1 C7 57 0D 3C 3D 82 ....D...K..W.<=.
)000D010  F1 9A 69 33 0E 93 DB EF 94 03 F9 12 6B 23 DD 7F ..i3........k#..
)000D020  32 4F 32 C2 99 9A 14 7A DE 5B C8 CD 56 CB 6F 03 2O2....z.[..V.o.
)000D030  E4 33 37 76 65 8D 78 30 EE 33 DB 2C 89 F5 6D 19 .37ve.x0.3.,..m.
)000D040  EE 33 DB 2C 89 F5 6D 19 C5 9A 08 8B 9B EF 82 EF .3.,..m.........
)000D050  95 B4 DA BA 33 D5 7A 6C AC 74 96 07 00 D5 DA EE ....3.zl.t......
)000D060  EE 33 DB 2C 89 F5 6D 19 E4 32 DD B7 ED EB 15 A1 .3.,..m..2......
)000D070  C5 BA 57 C3 9A AE 95 2C EE 33 DB 2C 89 F5 6D 19 ..W....,.3.,..m.
)000D080  EE 33 DB 2C 89 F5 6D 19 98 DC E3 EB A8 4D 71 62 .3.,..m......Mqb
)000D090  E8 F4 47 FF FB BC 07 9E E8 B2 97 AC F6 64 BF EC ..G..........d..
)000D0A0  EE 33 DB 2C 89 F5 6D 19 13 17 0B A2 2C D6 E8 AB .3.,..m.....,...
)000D0B0  77 73 CD C4 5D B0 80 9F B7 9A B3 AD D2 F2 61 FA ws..].........a.
)000D0C0  54 9D 57 DC 9F 6D CB F2 10 69 06 D1 5F 4C 00 AD T.W..m...i.._L..
)000D0D0  A6 2C 7F 82 5A E3 83 B8 EE 33 DB 2C 89 F5 6D 19 .,..Z....3.,..m.
)000D0E0  EE 33 DB 2C 89 F5 6D 19 1C 21 27 D6 1D 8C 0A 1C .3.,..m..!'.....
)000D0F0  71 37 2D 18 EE B9 26 35 6D 3E CF 53 86 9F D2 C8 q7-...&5m>.S....
)000D100  46 E0 1F 6E C6 B2 83 20 67 21 EA D1 97 DB 1A 46 F..n... g!.....F
)000D110  55 E9 52 A5 32 FA 49 1C 9F 51 FC 63 06 59 C0 FA U.R.2.I..Q.c.Y..
)000D120  99 02 3D F7 4D 0C 99 75 3A 07 09 10 88 BF B1 C1 ..=.M..u:.......
)000D130  C5 78 89 FA 35 21 69 88 EE 33 DB 2C 89 F5 6D 19 .x..5!i..3.,..m.
)000D140  36 61 92 D0 D1 EB D9 BA 99 FE 9C 0F 60 46 EA 47 6a..........`F.G
)000D150  EE 33 DB 2C 89 F5 6D 19 B0 FD 0F 30 67 E6 B4 79 .3.,..m....0g..y
)000D160  EE 33 DB 2C 89 F5 6D 19 EE 33 DB 2C 89 F5 6D 19 .3.,..m..3.,..m.
)000D170  EE 33 DD 2C 89 F5 6D 19 EE 33 DD 2C 89 F5 6D 19 .3.,..m..3.,..m.
)000D180  EE 33 DB 2C 89 F5 6D 19 EE 33 DB 2C 89 F5 6D 19 .3.,..m..3.,..m
```

If you are running PSQL v13.30 or above and need an even stronger algorithm, then consider upgrading your database files to the v13.x file format, where a file with a long owner name will be encrypted with a true AES-192 algorithm.

If you have Zen v14 and use long owner names up to 24 characters, then you'll also get the AES-192 encryption algorithm, but if you have an owner name with a length from 25 to 32 characters, then you'll go one better with AES-256 encryption.

Trade-Offs When Using Owner Names

As with any security solution, there are tradeoffs to using owner names. First off, every time you want to access a file, you have to now provide the owner name, which gets tedious for the application support personnel. While each file can have its own unique owner name, most applications use tens or hundreds of files (if not more), and most developers elect to stick with the same owner name for every file, limiting the complexity. Further, many developers never change the owner name on the files, as there is a considerable amount of work (and system downtime) in doing so. Finally, applications must include owner names in the application, so they either need to hide the text or leave it possible for a hacker to extract the owner name from the application code itself.

WARNING: If you enable owner names and then forget your owner name, you may lose access to your data. Whatever you do, document your owner name and don't lose it!

Trade-Offs When Using Long Owner Names

If you elect to use LONG owner names, then you do need to be aware of some additional restrictions. If you are using a v13.x file format but enable AES-256 encryption with an owner name of 25-32 characters, then the file will not be able to be opened in any version prior to v14.

If you elect to use any long owner names and the v13.x file format, you must have a minimum of v13.30 to be able to open the file. Attempting to open a v13 file on v13.21 or older will fail with an error.

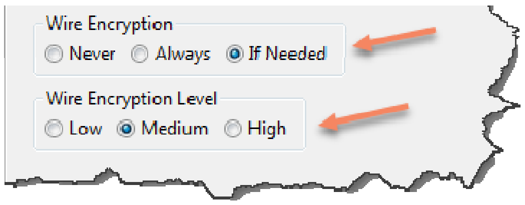
Across-the-Wire Encryption

Across-the-Wire Encryption was added in Pervasive.SQL V8.5 and serves to protect **data in motion**, encrypting the data as it is being sent out across the network wire from the client to the server. This feature can prevent someone with a network analyzer from capturing traffic and seeing database information in-flight.

Encryption is currently available in three levels, Low (40-bit), Medium (56-bit), and High (128-bit) and is implemented via the public-domain Blowfish encryption algorithm. Of note is the fact that there is no performance difference between the encryption levels, so it is unclear why one would ever want to select a level lower than "High".

Enabling and Disabling Across-the-Wire Encryption

You enable wire encryption through two three-state values, one on the Zen/PSQL Engine and one on the Zen/PSQL Client. Start the **Control Center** and select either *Configure Local Engine* (on the engine) or *Configure Microkernel Router* (on the workstation), and go to the *Access* tab. There, you will see two settings:


Wire Encryption
Never Always [✓ If Needed]
Wire Encryption Level
Low [✓ Medium] High

The default for *Wire Encryption* on both sides is "If Needed", which means that encryption will NOT be used unless this setting is changed. If you set the server to "Always", then every connection to that server will be encrypted. If you set a workstation to "Always", then every connection from that client will be encrypted. The default for *Wire Encryption Level* is "Medium", though we recommend changing this to "High" to use 128-bit encryption.

Trade-Offs When Using Wire Encryption

Since every packet will need to be encrypted before transmission and decrypted upon receipt, enabling wire encryption will definitely slow down your overall system throughput. While exact overhead will depend on the CPU speeds of the client and server, as well as total amount of traffic being transferred and server-side workload, seeing degradation of 10-20% may not be unusual.

Another sacrifice when using encryption is the loss of any ability to capture network data across the wire and debug applications based on that data. You'll be able to see that communications is occurring, but you won't be able to see SQL queries or Btrieve records going back and forth. This can make troubleshooting the application much more difficult.

WARNING: There was a bug in the ODBC layer of some versions that caused extremely slow communications to occur if you enabled both Wire Encryption and PARC (Auto-Reconnect) at the same time. If you are running an older release and experience performance problems with SQL calls, please patch your database engine or disable either PARC or encryption.

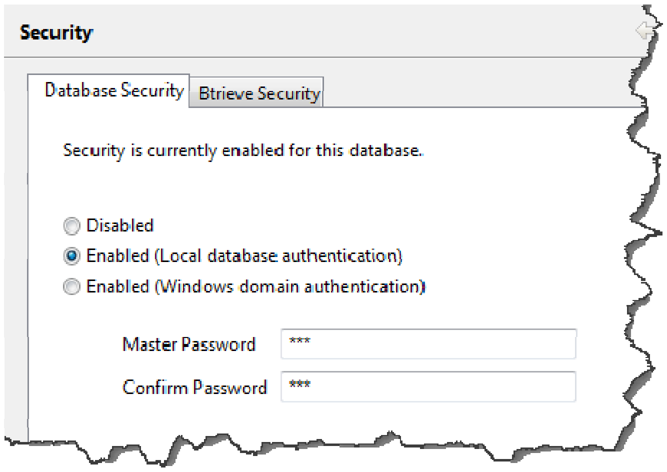
SQL Authentication and Access Control

The security feature is applicable to anyone who has valid data definition files (DDF's) and creates a SQL Database within the Control Center for their database. Once a named database is created to link the DDF's to the data files, anyone with a Zen/PSQL Client and an ODBC-based application can access the data. By default, every user on the network can not only see the data, but they can insert, update, and delete any record they want, too. Talk about scary!

SQL-level security is therefore the next step towards protecting your data. When SQL security is enabled on a database, the SQL/ODBC interface will require valid authentication in order to connect to the database engine. Further, you can grant specific rights to each user, controlling whether they can see data, change data, or delete data.

Enabling and Disabling SQL Security

SQL Security is enabled most easily through the **Control Center** by right-clicking on the database name and selecting *Properties*, then going to the *Security* page, and then to the *Database Security* tab:



Security is currently enabled for this database.

Disabled [✓ Enabled (Local database authentication)] Enabled (Windows domain authentication)

Master Password ***

Confirm Password ***

(The screenshot here is from PSQL v13.10 and higher. If you see only two options, then you are probably running an older engine version.)

When you select the **Enabled** radio button, you will also provide the Master password. The Master password is exactly that – it is the password to be used for the user name “Master”, which is the default administrative user in PSQL. Note that both the user name AND password are case sensitive, so take special care when entering your password, and please do not forget it! You’ll need to provide the Master password if you ever want to disable security again.

After you click OK here, your database will be secured, and a login will now be required for all new SQL connections. You will likely want to create user accounts, as well as optional groups, and then grant rights to the database or individual tables to those accounts. These changes can be made via the GUI in the PCC or through SQL commands like GRANT and REVOKE. In the interest of brevity, we won’t cover the myriad of security options here, but will instead just refer you to the online documentation for more details on how to do this correctly.

Disabling security is also fairly easy – just log into the database with the Master password, right-click the database in the Control Center, go to the *Properties* page, the *Security* tab, and change the radio button back to **Disabled**.

Trade-Offs When Using SQL Security

If you have a Btrieve-level application (i.e. one that works exclusively at the navigational layer) instead, the SQL security will be completely transparent. In fact, those applications will never even know that SQL security has been enabled. This makes it possible to retrofit SQL security onto a large number of older Btrieve applications without worry.

However, enabling SQL security means that **every single SQL/ODBC connection will now have to authenticate**. This can be built into the connection strings used by applications, or the application may present a separate login dialog for the user to enter credentials. Poorly-written applications may simply refuse to connect, as they may provide no facility for entering the username and password! If this is your case, you’ll need to contact the application developer for help in enabling security.

A big potential problem can be created when developers push out an application update, especially if they are not aware that you have enabled security. Often, developers will simply replace your existing (old) DDF’s with a completely new set of DDF’s for the new application and data structures. Suddenly, your secured database is left with no security controls at all! This can either disable your SQL security (leaving your system wide open again), or it can erase all of your carefully-defined users, groups, and access rights, making you go through the definition process all over again. (This is a good reason to use scripts to set up groups, users and access rights.)

Another potential trade-off is the need to maintain a completely independent security subsystem within the database. For organizations with lots of users, this can take quite a bit of time to set up, as well. Further, there is no password complexity requirement, and no way to force password changes on a periodic basis, so this solution may not meet your auditing requirements. In fact, the user name and password may be limited to only 8 characters (for v1 metadata, but 128 characters for v2 metadata). Luckily, the operating system has many of the additional control features you are looking for, so if you are running v13.10 or above, you can leverage the third option shown in the dialog above, domain authentication (discussed next).

Domain-Authenticated SQL Security

This new feature was added to v13.10 to eliminate the headaches caused by maintaining a separate security subsystem within the database. If you select this option and attempt a SQL login, then there is no separate login prompt required. Instead, the current user's login name and password are pulled from the operating system and submitted to the PSQL engine. The engine then checks to see what AD groups the user is a member of. One (and exactly one) of these groups should match a SQL security group defined within the database, which has specific rights granted to it. This user will then inherit all of the rights of that one group. As this feature completely eliminates the need for user accounts, the Control Center will even hide this from the UI, too.

Enabling and Disabling Domain-Authenticated SQL Security

See the previous section to learn how to configure a database for domain authentication. You will also need to create groups in the database (which can be done with the Control Center GUI or through CREATE GROUP statements in SQL) that match the names of the groups defined in Active Directory, as well as rights for those groups. You should then ensure that each AD user account is a member of one, and only one, of these groups.

While it is not required that you use different group names for AD security and SQL security, doing so may be easier if your AD groups do not align perfectly with your application groups.

Trade-Offs When Using Domain-Authenticated SQL Security

As indicated, you must ensure that users are only a member of EXACTLY ONE of the available groups, to avoid confusing the engine. Otherwise, trade-offs are pretty much the same as the local SQL security option.

Advanced Btrieve Security

The final mechanism for securing your databases is the most complicated to implement, but it also provides the most secure environment. As its name implies, this advanced layer of security is designed to help secure the Btrieve database files, which are otherwise not protected when enabling SQL-level security.

The key to understanding Advanced Btrieve Security is in understanding two concepts:

- **Authentication** is the process of proving who a user is to the environment. This is commonly done today via a username and password, though biometrics at the OS level are starting to get more popular.
- **Authorization** indicates what access rights a user actually has, such that users are only able to access the data which has been explicitly granted to them.

These two concepts are combined in three different combinations to provide three different advanced security mechanisms, which are:

Security Model	**Authentication**	**Authorization**
Classic	Operating System	Operating System
Database	Database	Database
Mixed	Operating System	Database

Let's see how each security model works in a bit more detail.

Classic Btrieve Security

Classic Btrieve security is what you get by default. This security model leverages the operating system to handle both authentication and authorization. When a user logs onto a Windows workstation, he or she is given a Security ID (SID) that indicates the specific user account. This SID is submitted to the database engine when a connection is made and stored by the engine for tracking purposes. Every time a database file is opened at the Btrieve level, the database engine compares the access rights of the SID against the access rights of the file, and either allows (or prevents) access to the file.

The advantage to Classic security is that it is easy to understand, and it has been the default model since the 1980's. If a user can see and open the database file at the OS level, then he or she can open it at the database level. Of course, the reverse is also true – if the user must be able to open the file within the database application, then he must also have rights to the file at the OS level – which leaves the file open to being read, updated, or possibly even deleted. This is great if you trust your users, but a disgruntled user can easily cause lots of havoc in a system with OS rights to the raw files, and a corporate spy with even read-only access can easily extract your entire customer list with a simple file copy command!

Important Note: This description applies to the Server engines only. The Workgroup Engine does not enforce any operating system rights at all. If you are using the WGE, then any user who knows the location of the file can access it!

Database Btrieve Security

With Database security, we can completely remove the operating system rights from the mix. Instead of authenticating within the OS, we instead authenticate separately to the database engine. Database security requires that SQL-level Security be enabled, and that users (and optional groups) be configured properly with appropriate rights. Applications can then either request a username/password combination to submit to the engine (via the Btrieve Login command or through a URI), or the PSQL Client can pop up a separate dialog to obtain access credentials. These credentials are then compared against those stored in the database itself, and the account privileges are used to control access rights from there.

The primary advantage to security method is that there is NO requirement for operating system rights, so the rights to the underlying files or folders can be completely revoked, securing the data from OS-level access, and thus limiting the ability to copy off the files or for malware to encrypt them. On the other hand, you must set up and maintain SQL security, and every time you add a new user to the OS, you have to also remember to explicitly add them to the database, unless you are using generic credentials from within the application. Another downside is that the SQL security implementation does not provide for password complexity rules or mandatory password changes every X days.

Mixed Btrieve Security

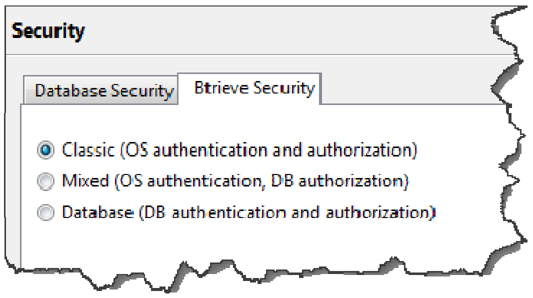
Mixed security is not a combination of the two above mechanisms, but rather is simply a mixture of using the OS for your authentication and the database for authorization. The OS is a superior authentication solution because it already allows for all of the password complexity and change features required by many auditors, and it can leverage more complicated biometrics quite easily. After you have authenticated to the OS, these credentials (i.e. the authenticated user name) are passed to the database for authorization purposes, and the SQL-layer security handles controlling access.

As with Database security, you can restrict or completely eliminate OS-level access rights to the database files and folders with this solution. Because the OS is handling authentication, you can use the stronger requirements (controlled at the OS layer) for password complexity and change rules. The only requirement is that you set up a user name in the SQL security layer that exactly matches each user in your OS security environment.

Important Note: As indicated in the section on Classic Security, the Workgroup Engine does not enforce any operating system rights at all. If you are using the WGE, it is not possible to use Mixed Btrieve Security.

Enabling and Disabling Advanced Btrieve Security

If you decide that you want to use Advanced Btrieve Security in a mode other than Classic, your first task is to decide which method is best for you, Database or Mixed. Then, you need to enable SQL-level security on the database and configure the groups, users, and access rights within the database environment. Finally, you go back to the PCC, log into the database as the Master user, then right-click on the database name, select *Properties*, select the *Security* page, and then select the *Btrieve Security* tab:


Security

Database Security | Btrieve Security

[● Classic (OS authentication and authorization)]
○ Mixed (OS authentication, DB authorization)
○ Database (DB authentication and authorization)

Once you are here, you can switch to either Mixed or Database security. Note that once you have selected one of these security models, you will need to first disable SQL security before you can switch to a different model.

Trade-Offs When Using Advanced Btrieve Security

While it sounds like this is the perfect way to lock down your system, there are some caveats that you must be aware of. There's actually so many that we need a numbered list to handle them all!

1. Because SQL Security is a requirement, you must first be sure that all applications that access the database via SQL are able to provide SQL logon credentials. For some applications, this is only a matter of adding "UID=<user>; PWD=<password>;" to the connection string. However, some applications may not support this out of the box.
2. You will need to set up user names and access rights within the SQL environment. If you take a shortcut and assign rights at the database level, then users may have access to files that they shouldn't have. If you assign rights at the individual table level and the developer adds new tables to the database, your application may break until you can add rights to the new tables to each user/group that needs access.
3. Sometimes, developers pushing out a new application release will replace your DDF's, which will break the entire security model of the application, since users will no longer be available in the SQL Security layer.
4. Some applications store non-database files in the database data folders. This can include files like INI or configuration files, simple text files, exported CSV or XML data, or even temporary flat files used to store intermediate results for a report. If you remove OS-level access rights to an application like this, then you risk breaking that application.
5. Some applications access database files using OS-level commands, such as DELETE, RENAME, and so on. If these accesses only occur at known times (such as a major application update), then you can take care to use an administrative login that does have rights to perform these tasks. However, if an application creates a temporary database file for collimating reporting data and then attempts to delete that file using an OS command at the end of the process,

this last operation may fail. (Developers can use the DeleteFile and RenameFile functions within the database to handle this, but many do not.)

6. If applications do not use the Btrieve Login command to link to a database, then another method of linking a data set to a database name is needed, such as a URI or by using IDSHOSTS redirection. Further, if the user ID and password are not provided to Database Security, an extra login dialog box may be needed to authenticate properly, which can be disruptive to the users.
7. There is always a risk of an over-zealous system administrator removing access rights to a folder from everyone, such that nobody can access the files any more, making it even impossible to back up the data set!
8. Finally, troubleshooting an environment like this is much more complicated, as only specific (administrative) users will have the rights to see the database folder and make certain changes.

What Should I Do To Secure MY Data?

There is no one right answer to this question, of course. Adding security will always require additional administrative effort to implement and maintain, especially as applications and databases grow and change, and extra security will always act to slow the access to critical data. However, security is becoming more and more important in the face of legislation and the latest “best practices” in any data-centric environment.

The best thing we can say at this time is simply to “give your database the protection it warrants.” A simple reporting database may not require much in the way of security, but an ERP system storing invoices, credit card numbers, and other such financial data should certainly be protected with one or more of these mechanisms.

If you still can't get it to work, contact Goldstar Software and let us work with you to help! Please note that this may be a billable support call if you have already used up your free support time.