Db2 V12, Raising the Security Bar

A computer network diagram with icons and symbols

AI-generated content may be incorrect.

Db2 V12 is raising the bar on security, delivering advanced new features, security focused default behaviour changes and removal of older features not up to modern standards. With this blog, we are launching a new series highlighting the key changes we’ve made to achieve these goals. In this first post, we are going to focus on the changes in V12 that may impact your operations, and how they are designed to enhance your security stance. Future posts will dive more in-depth into new functionality such as trust procedures for establishing a trusted context, audit exceptions, and alternative masking semantics.

Major releases are a time for housekeeping. As part of this, you often allocate a little more time to adopt it than a modpack, and perhaps a little more time to reading What’s New and What’s Changed documentation. For us, major releases are the best time to introduce impactful security changes with the highest change, as we expect they will be noticed and acted upon prior to the upgrade. Db2 v12 has an extensive list of such changes that we are about to dive into.

We will start our journey with some of the more subtle changes that can cause difficult to diagnose errors as the features themselves have not changed, but some default behaviour has, all in the name of improving your security posture of course. We will follow that with a deep dive into FIPS 140-3 related changes, finishing with some important to know about discontinuations.

# Default Behaviour Changes

## Default GRANT behaviour

Many versions ago Db2 introduced the SECADM, DATAACCESS and ACCESSCTRL authorities, carving up the SYSADM and DBADM authorities into multiple pieces inside the database, in a project known as separation of duties. Prior to that the DBADM was able to do just about everything inside the database, including selecting from any table and performing most grants. In order to provide backwards compatibility, the GRANT DBADM statement was modified to also grant DATAACCESS and ACCESSCTRL authorities. That syntax has stuck around until V12, where the raised security bar has highlighted this potentially insecure behaviour. I say potentially because it is well documented (and we all read every page of the documentation, right?), and there is syntax that says not to document. But it nonetheless could catch the wary off guard. To meet the V12 bar, we are removing the default grant of DATAACCESS and ACCESSCTRL when granting DBADM. The full syntax is still supported, only the default when no option is specified is changing.

You can compare the syntax here:

V11.x:

  .-WITH DATAACCESS----. .-WITH ACCESSCTRL----.

+-DBADM--+--------------------+---+--------------------+--

        '-WITHOUT DATAACCESS-' '-WITHOUT ACCESSCTRL-'

V12.1:

  .-**WITHOUT** DATAACCESS-. .-**WITHOUT** ACCESSCTRL-.

+-DBADM--+--------------------+---+--------------------+--

        '-WITH DATAACCESS----' '-WITH ACCESSCTRL----'

No changes to existing grants performed, this change in behaviour only affects newly issued GRANT DBADM statements. This has the potential to affect existing applications, so a registry variable “db2set DB2\_ALTERNATE\_AUTHZ\_BEHAVIOUR=DBADM\_ADDTL\_AUTHS” can be used to revert the behaviour.

Another authorization behaviour change is to stop granting numerous database privileges to the PUBLIC during a CREATE DATABASE. These privileges were available to everyone to promote an open and easy to use database. Db2 is more than 30 years old, and there were different priorities then.

Starting in V12, the following privileges are no longer granted to PUBLIC:

* CONNECT
* IMPLICIT\_SCHEMA
* CREATETAB
* BINDADD
* USE on USERSPACE1
* CREATEIN on SQLJ and NULLID schemas

As the with the DBADM change above, no existing grants are modified. This change only affects new databases that are created. The registry variable “db2set DB2\_ALTERNATE\_AUTHZ\_BEHAVIOUR=PUBLIC\_DBCREATE” can be set to undo this behaviour if it breaks applications that cannot be modified. Note that you can set multiple values for this registry variable separated by commas, so for example db2set DB2\_ALTERNATE\_AUTHZ\_BEHAVIOUR= DBADM\_ADDTL\_AUTHS,PUBLIC\_DBCREATE” enables both behaviours.

## Hostname Validation

Hostname validation is a TLS related feature where the Db2 client will validate that the hostname it is connecting to is contained within the certificate returned from the server after validating that the certificate is signed by a trusted Certificate Authority. This ensures that someone is not masquerading as the Db2 server and intercepting the client’s traffic somewhere on the network. The attacker may be able to present a properly signed certificate, but the contained hostname will not match the connection attempted, as the attacker would require control of the server to prove their ownership. In v11.5.6 this feature was introduced but off by default, as there was very real risk it would break customers connections when their certificate was not created correctly. Db2 V12 clients are now enforcing hostname validation by default as it ensures the client is really communicating with the server it expects. There is a great IDUG Blog on this feature which goes into much more detail <https://www.idug.org/news/hostname-validation-in-db2>.

# Preparing for FIPS 140-3

Federal Information Processing Standards (FIPS) 140-3 is a set of security requirements for cryptographic modules published by the United States government. FIPS 140-3 is typically required for use within the United States Federal Government agencies, although is also sometimes mandated by various industries or individual customers. Db2 V12.1.0.0 uses a version of “GSKit”, Db2’s internal cryptographic module, which was certified against FIPS 140-2, and we are waiting for a FIPS 140-3 certified version for inclusion in a future modpack. However, the security requirements mandated by FIPS 140-3 are enhanced from the past, and Db2 v12 includes the impactful changes now in preparation for future adoption of the new module. This is continuing the theme of taking advantage of the major release to introduce the most impactful changes on daily operations.

## FIPS Modes

As you know, Db2 has long shipped with a “FIPS” mode on/off switch as part of the DB2AUTH registry variable. This is now enhanced to support three modes, NOFIPS, FIPS and STRICT\_FIPS.

**NOFIPS**: Db2 does not enforce any FIPS 140-3 requirements. The cryptographic module used by Db2 will be the latest version available, which may include enhancements and performance improvements that are not yet included in the FIPS 140 certified version, which tends to lag a little behind due to the certification timeframe.

**FIPS**: In this mode, which is the default, Db2 attempts to honour the spirit of FIPS requirements, but not necessarily the law, to provide the most compatibility. For example, in response to security vulnerabilities in the past, fixes have been available in the non-FIPS certified cryptographic module far in advance than the FIPS certified module. Use of FIPS mode allows Db2 to make use of that non-FIPS certified module to pick up that specific feature instead of disabling some functionality. Legacy Db2 behaviour that does not meet current FIPS 140-3 requirements is grandfathered in and allowed.

**STRICT\_FIPS**: Db2 strictly adheres to the FIPS 140-3 requirements, including preventing the use of certain Db2 features that do not meet the requirements. Only the FIPS 140 certified version of the cryptographic module may be used.

The mode can be changed by adjusting the DB2AUTH variable. If nothing is set, then FIPS mode is the default. You can not explicitly set this value. To use NOFIPS or STRICT\_FIPS, issue the following command:

db2set DB2AUTH=NOFIPS | STRICT\_FIPS

As the DB2AUTH variable can contain multiple values, how you unset NOFIPS/STRICT\_FIPS to return to the default FIPS mode depends on whether other values are also set.

If you have multiple values, such as “DB2AUTH=NOFIPS,OSAUTHDB”, then you can simply remove the NOFIPS with

db2set DB2AUTH=OSAUTHDB

If you only had the single values, such as “DB2AUTH=NOFIPS”, then you can supply an empty value, as in:

db2set DB2AUTH=

For more information about the different security modes, you can check out <https://www.ibm.com/docs/en/db2/12.1.0?topic=compliance-security-modes-in-db2>

## RSA Key Exchange Cipher-suites Disallowed

FIPS 140-3, via the NIST SP800-132aR2 standard, disallows the use of RSA as a key exchange mechanism in TLS 1.2, due to several issues such as the lack of forward secrecy. TLS 1.3 has completely removed any cipher suite that uses RSA key exchange. Unless you have explicitly specified a cipher suite utilizing RSA key exchange, then you are unlikely to be impacted by this change, as by default there are many choices available. You can tell if this applies to you by examining the SSL\_SVR\_CIPHERSPECS database manager configuration parameter. The cipher would look something like TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA25. Note that RSA as an authentication type is perfectly valid, it is only the key exchange which has issues. For example, TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 uses elliptic curve Diffie-Hellman (ECDHE) for key exchange, and RSA for authentication, so it is fine. FIPS and STRICT\_FIPS mode block the use of RSA key exchange in TLS 1.2.

## Strict Signature Algorithm Checking

As we will discuss in more detail below, the use of SHA1 as a signature algorithm has been removed from use for server certificates. However, since root CA certificates often have very long lifetimes, until recently up to 20 years, use of SHA1 for their signature is still allowed. However, with “strict sigalgs” enabled, their use is not allowed, according to NIST SP 800-131aR2. Therefore, when STRICT\_FIPS mode is enabled, use of SHA1 for any certificate is not allowed. Remediation requires creating a new certificate. FIPS and NOFIPS continue to allow the use of SHA1 for root or intermediate CA certificates (but not for server certificates).

You can determine the type of algorithm used for the certificate by issuing the following command with the label of the root CA certificate. Repeat it for any intermediate certificates:

gsk8capicmd\_64 -cert -details -label <rootcalabel> -db < keystorefrom-SSL\_SVR\_KEYDB> -stashed | grep -I signature

Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)

If you see SHA1 then you will have errors in STRICT\_FIPS mode. In the example above SHA256 is being used and thus has no problems.

Note: while unlikely to be in use for a root CA certificate, MD5, as an even older and weaker algorithm than SHA1, falls under the same behaviour as above with regards to being disallowed in STRICT\_FIPS mode.

## Use of CHACHAPOLY20 and X25519

The very popular encryption algorithm CHACHAPOLY20 and elliptic curve X25519 are known for their small size and good performance. They are well regarded and widely adopted algorithms; however, they are not FIPS certified so their use is not allowed when running in STRICT\_FIPS mode. The following diagram shows when their use is allowed, and whether it is enabled by default or needs to be configured through SSL\_CIPHERSPECS if you wish to use it.

|  |  |  |
| --- | --- | --- |
| CHACHA/X25519 usage | **Client** | **Server** |
| STRICT\_FIPS | Not supported | Not Supported |
| FIPS | Enabled by default | Configurable |
| NOFIPS | Enabled by default | Enabled by default |

### Extended Master Secret

This optional feature of TLS 1.2 and built-in feature of TLS 1.3 protects against Triple Handshake person-in-the-middle attacks. FIPS 140-3 requires the use of Extended Master Secret.

**Critical** – supports and requires use of EMS. Connections without EMS will fail.

**Advisory** – supports use of EMS but not required. Connections without EMS will continue to work

The following explains what versions of Db2 support EMS:

**V11.5 C clients** – N/A, does not support EMS

**V11.5 Java clients** – advisory\*

**V11.5.9 Server** – advisory

**V11.5.9 HADR** – N/A does not support EMS

**V12 C clients FIPS/NOFIPS** – advisory

**V12 C clients STRICT\_FIPS** – critical

**V12 Java clients** – advisory \*

**V12 Server, HADR FIPS/NOFIPS** – advisory

**V12 Server, HADR STRICT\_FIPS** - critical

\* The java behaviour listed is the default. It can be configured using the jdk.tls.useExtendedMasterSecret and dk.tls.allowLegacyMasterSecret JDK configuration options.

Given the above information, we can see that if TLS 1.2 is used between an 11.5 client and a v12 server running in STRICT\_FIPS mode, that the connection will fail because the server requires the use of EMS, but the client does not support it. The recommended approach in this scenario is to use an 11.5.9 client, as both that client and the server support TLS 1.3 where EMS is a standard part of the protocol and always used.

## SERVER\_ENCRYPT and ENCRYPT UDF

The SERVER\_ENCRYPT authentication type makes use of the Diffie-Hellman key exchange protocol for the client and server to determine a secret cryptographic key over an insecure network. The ALTERNATE\_AUTH\_ENC database manager configuration parameter can configure the use of AES encryption. However, even still the size of the Diffie-Hellman key that is exchanged is less than that allowed within NIST SP800-131aR2. As such, the use of SERVER\_ENCRYPT authentication is blocked in STRICT\_FIPS mode, but its use is allowed in NO\_FIPS or FIPS mode. TLS is the recommended technology for encrypting network communications between client and server. To help support the migration to TLS, Db2 now considers SERVER authentication over a TLS encrypted connection to be compatible with SERVER\_ENCRYPT authentication.

Similarly, the RC2 encryption algorithm that is used as part of the ENCRYPT user defined function (UDF) is not considered secure, and is also blocked in STRICT\_FIPS mode, but allowed in FIPS or NOFIPS mode. Decryption is allowed in all modes for access to existing data. Note the ENCRYPT and DECRYPT UDFs are deprecated, and database level Native Encryption is recommended instead.

## Summary of FIPS behaviour

The following chart summaries which functionality is available or restricted depending on the FIPS mode that is being used.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **STRICT\_FIPS** | **FIPS** | **NO\_FIPS** |
| SERVER\_ENCRYPT | ⅹ | ✔️ | ✔️ |
| ENCRYPT UDF | ⅹ | ✔️ | ✔️ |
| RSA Key Exchange | ⅹ | ⅹ | ✔️ |
| CHACHA\_POLY Cipher | ⅹ | ✔️ | ✔️ |
| x448 and x25719 key exchange groups | ⅹ | ✔️ | ✔️ |
| Signature algorithms weaker than SHA256 in root and intermediate certificates | ⅹ | ✔️ | ✔️ |
| TLS 1.2 connections without extended master secret | ⅹ | ✔️ | ✔️ |

# Discontinuations – Encryption

## TLS Versions 1.0 and 1.1

Encryption is an area of rapid technological improvements, advancing the state of the art as well as removing older and insecure pieces. An example of such advancement is in the use of Transport Layer Security (TLS) for securing communications between two parties. Db2 has supported the latest version TLS 1.3 since 11.5.8. However, until v12, Db2 has continued to support the out of date and insecure versions of TLS 1.0 and 1.1. We have taken the major release of Db2 12 to remove these from the Db2. Db2 has supported TLS 1.2 for more than a decade (Db2 9.7 FP9, 10.1 FP 4, 10.5 FP 3, 11.1) and there is no reason to make use of such insecure versions. Unless you have explicitly configured these versions in the SSL\_VERSIONS database manager configuration (DBM CFG) parameter, then there should be little for you to do, the default in Db2 V12 is TLS 1.3 with support for TLS 1.2. One important addendum: the minimum RSA key size for your server certificate is now 2048 bits, you will need to validate the size of your certificate to ensure it is large enough. The minimum key size for EC certs has not changed from 256 bits.

You can gather the certificate size at the server with the following commands:

Obtain the SSL label and keystore file from the DBM CFG:

db2 get dbm cfg | grep SSL\_

Look for the SSL\_SVR\_LABEL and SSL\_SVR\_KEYDB values. Then issue:

gsk8capicmd\_64 -cert -details -label <labelfrom-SSL\_SVR\_LABEL> -db <keystorefrom-SSL\_SVR\_KEYDB> -stashed

Look for

Key Size : 2048

## SHA1 Signature Algorithm

With regards to digital signatures, the SHA1 hashing algorithm is considered insecure, with collisions having been produced using significant, but everyday less significant, computing power. And to complete this, rainbow tables of short texts are readily available to anyone. For these reasons, it was about time that the use of SHA1 as a signature algorithm for server certificates was removed, and Db2 V12 was the right time to do this. If SHA1 is being used for your server certificate, it will need to be updated.

You can determine the type of algorithm used for the certificate in the same manner used to get the key size above:

gsk8capicmd\_64 -cert -details -label < labelfrom-SSL\_SVR\_LABEL > -db < keystorefrom-SSL\_SVR\_KEYDB > -stashed | grep -I signature

Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)

If you see SHA1 then you will need to recreate your certificate. In the example above SHA256 is being used and thus has no problems.

## 3DES Native Encryption

The 3DES encryption algorithm is no longer considered secure, due to its small block size, limited effective key strength, vulnerability to modern attacks like Sweet32, and official deprecation by standards bodies like NIST, and should not be used for encrypting data anymore. In the case of Db2, it was used alongside AES as one of the algorithms for encrypting data at rest as part of Db2’s Native Encryption. And you can see the theme here, it had to go to, so starting in V12 you are no longer able to encrypt a new database, or restoring into a new database, using the 3DES algorithm. Rest assured though that any data you already have encrypted with 3DES, be it the database or a backup, is fully accessible. We have not removed the ability to decrypt data using 3DES, but we strongly recommend that you move to adopt the more secure algorithms Db2 supports, like AES. We will get back to this point a bit later on.

Let’s walk through what is supported—and what is not. Recall that Db2 treats restoring a database on top of an existing database slightly differently than when restoring a database to a location without the database (a new system or you have dropped that database prior to the restore).

The following scenarios are supported

* ACTIVATE DATABASE when it is encrypted with 3DES
* Restoring from a backup encrypted with 3DES
* Restoring on top of an existing database that is encrypted with 3DES
* Migrating a database from V11.x to V12 that is using 3DES

The following scenarios are not supported

* Creating a new database using 3DES
* Restoring into a new database using 3DES
* Create a backup using 3DES
  + By default, the backup uses the same encryption algorithm as the database, so if your database is using 3DES, you will need an option to create the backup with AES
    - ENCROPTS (database configuration or backup command) to specify AES

Like we said, AES is the appropriate algorithm to use as a replacement, and it should perform better as well! To change the algorithm for the database, you need to follow the same steps as when enabling native encryption from the start, that is take a backup, drop the database, and then do a restore specifying AES. An existing IDUG blog details steps for using HADR to enable encryption with minimal downtime, which would apply to changing algorithms as well: <https://www.idug.org/news/encrypting-a-db2-database-with-minimal-downtime-using-hadr>

# Discontinuations – Authentication

## DATA\_ENCRYPT Authentication

The DATA\_ENCRYPT authentication type has been very popular as little setup is required to enable it. It was introduced in a similar timeframe as TLS to encrypt Db2 client/server communications. However, with the explosive growth of TLS, it eventually became clear that was the industry standard, and it should be Db2’s strategic direction. As a result, there have not been any updates to DATA\_ENCRYPT and it continues to use the very insecure DES encryption algorithm. Time for it to go, it doesn’t come close to clearing the bar set for V12.

## CLIENT Authentication

CLIENT authentication is a mechanism where the server can be configured to trust certain clients to have performed the authentication, rather than the server doing it. This can give the appearance of a single sign-on, as a password does not need to be sent to the server. The Db2 server can trust everyone, only clients it knows have an underlying authentication system, and Db2 for Z or i (called DRDA clients). Outside of very special cases (perhaps two air-gapped computers), none of these are secure. In all cases you are trusting the client – basically a remote network connection – and you have no real method of determining who that client is. Someone could connect a malicious laptop to the network with a Db2 client installed and any user they want defined on that laptop and connect to the database. CLIENT authentication is being removed to prevent accidental usage by the unwary. Almost. There are some very special cases that an alternative mechanism, such as Kerberos which securely provides single sign-on, is not appropriate. For those cases there is a registry variable that allows the use of CLIENT authentication. I’m not going to tell you what it is though, if it is important enough you can take the time to look it up, all the while taking the time to confirm the risk you are accepting. As mentioned, Kerberos is the closest in functionality.

# Conclusion

With adherence to modern standards and best practices comes some pain and work when we have fallen out of step with those practices. The changes made in Db2 V12 are intended get us back in step, however implemented in such a way that causes the least impact and incompatibilities to the widest number of users. Like we said, future posts will explore the newly introduced functionality in more detail, stay tuned for these! Until next time.

# About the Author

Greg Stager is the security architect for Db2 LUW at the IBM Toronto Lab. Greg has been a member of the Db2 security development team since 2000, where he has worked on all aspects of security within Db2, including authentication, authorization, auditing, and encryption. Greg is a primary contributor to the Db2 LUW CIS Benchmark, and a Certified Information System Security Professional (CISSP). Greg can be reached at gstager@ca.ibm.com

Cyrus Ng is a Software Developer on the Db2 Security team at the IBM Toronto Lab with a Bachelor's in Computing from Queen’s University. He has worked with various security features in Db2, such as TLS 1.3 support, hostname validation, and audit. Cyrus can be reached at cyrus.ng@ibm.com.