



The goal of this presentation is to take you through a quick overview of the security capabilities offered by Db2. We will be focusing (mostly) on built-in Db2 security functionality and not complementary capabilities offered by other products nor will we be looking at other important security areas such as dealing with compliance issues.

Since this presentation is only intended as an overview of Db2 security function, we will not be covering some very important areas in the security domain such as compliance or “hardening”.

These are the 4 main areas of security from a functional perspective.

Database security landscape

Authentication

Authorization

Auditing

Encryption

2

The goal of this presentation is to take you through a quick overview of the security capabilities offered by Db2. We will be focusing (mostly) on built-in Db2 security functionality and not complementary capabilities offered by other products nor will we be looking at other important security areas such as dealing with compliance issues.

Since this presentation is only intended as an overview of Db2 security function, we will not be covering some very important areas in the security domain such as compliance or “hardening”.

These are the 4 main areas of security from a functional perspective.



Authentication

Proving you are who you say you are

- Authentication is the process used by Db2 to validate that the credentials presented for an external user identity are valid and meant for use with the given user identity
 - Db2 relies on external 3rd parties to provide this validation
- The mechanism used for this validation is defined by the AUTHENTICATION database manager configuration parameter
 - All databases under the same Db2 instance use the same authentication mechanism
- Results of a successful authentication:
 - External User ID is mapped to a Db2 authorization ID
 - Any externally defined groups associated with the user are mapped to Db2 authorization IDs
 - Group membership is also defined outside of Db2

4

- Process by Db2 to validate that credentials presented for an external user identity are valid and meant for use with the given user identity
- Relies on 3rd parties to validate
- Mechanism defined by AUTHENTICATION database manager configuration parameter
- External User ID mapped to a Db2 authorization ID
- External groups associated with user mapped to Db2 authorization IDs

Authentication options

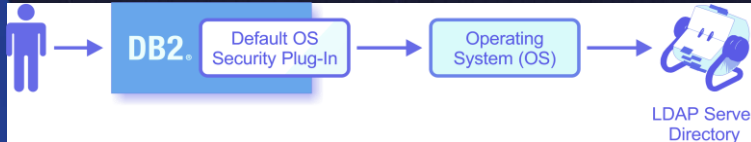
- Operating system (default)
 - User validation using a password
 - Group membership
- Kerberos
 - User validation using a Kerberos ticket
 - No group membership
- LDAP Plug-in
 - User validation using a password
 - Group membership
- Customized (via Plug-in)
 - User validation (e.g., GSS API) using different credential options
 - Group membership

5

- Default plugins, LDAP plugin use username and password from operating system
- Custom plugins can use different credential options
- Kerberos uses Kerberos to authenticate
- All authentication methods can be configured to provide group membership information except for Kerberos

Most popular? Transparent LDAP

- LDAP = Lightweight Directory Access Protocol (LDAP)
 - Standard protocol for accessing information at a directory server
 - Enables centralized authentication services for use across enterprise



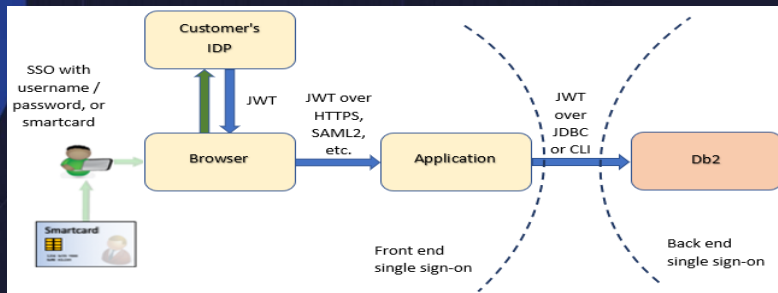
- The **transparent LDAP** approach integrates LDAP at the OS level which means both OS and Db2 authentication requests are satisfied by the same mechanism
 - Db2 authenticates users and acquires their groups via regular OS APIs
 - Enabled by setting DB2AUTH registry variable to "OSAUTHDB"

6

- LDAP = Lightweight Directory Access Protocol
- Transparent LDAP means Db2 authenticates via OS APIs

Single sign-on with JSON Web Token (11.5.4.0)

- Authentication without username/password via JSON Web Token (JWT)
- Identity Provider (IDP) signs the token that is later passed to Db2
- Db2 is configured to "trust" IDPs and use their public key to verify the token
- If the token is valid, Db2 uses the identity within the JWT for authentication



7

RFC 7519: a compact, URL-safe means of representing claims to be transferred between two parties

Db2 does not generate JWTs. IDP example: Gluu, Knox SSO

New srvcon_auth types: SERVER_ENCRYPT_TOKEN, GSSPLUGIN_TOKEN etc.

Example

- Sample JWT payload:

```
{  
  "name": "John Doe",  
  "issuer": "KNOXSSO",  
  "username": "admin",  
  "exp": 1516239022  
}
```

- On CLP:

```
CONNECT TO dbname ACCESSTOKEN <token> ACCESSTOKENTYPE JWT
```

- In JDBC using API

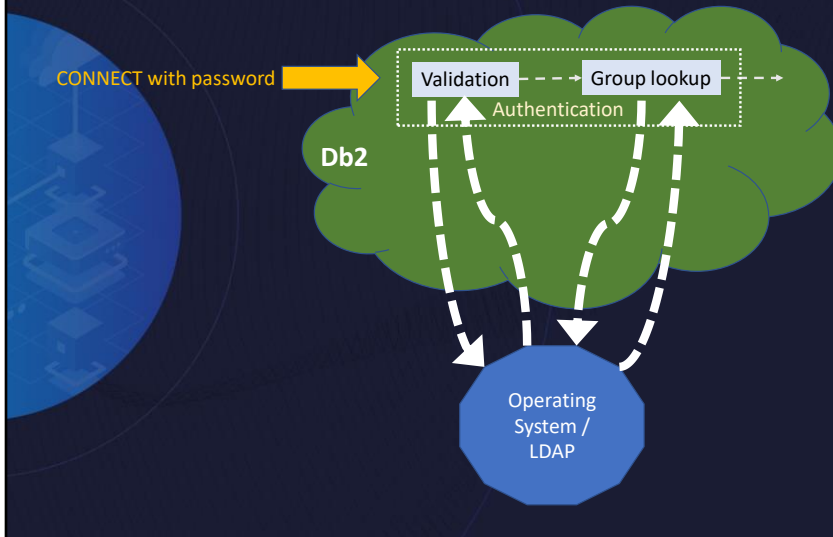
```
dataSource.setAccessToken( "<token>" );  
dataSource.setAccessTokenType( "JWT" );  
...  
Connection conn = dataSource.getConnection( );
```

Support for: CLP, CLI, JDBC .NET, Embedded SQL, CLI derived language support (e.g. Python, Ruby)

Authentication cache (11.5.3.0)

- Introduced to help improve performance for password-based authentication in the following scenarios:
 - Connections are of extremely short duration
 - Authentication “pipeline” gets overwhelmed
- Cache contains the results of successful authentication efforts and group lookup results
 - Results are only kept for limited amount of time (configurable)
 - Cache size is configurable by number of unique user IDs to be cached
- When enabled, incoming requests (and associated credentials) are compared to cached entries and, if match found, further authentication processing is bypassed

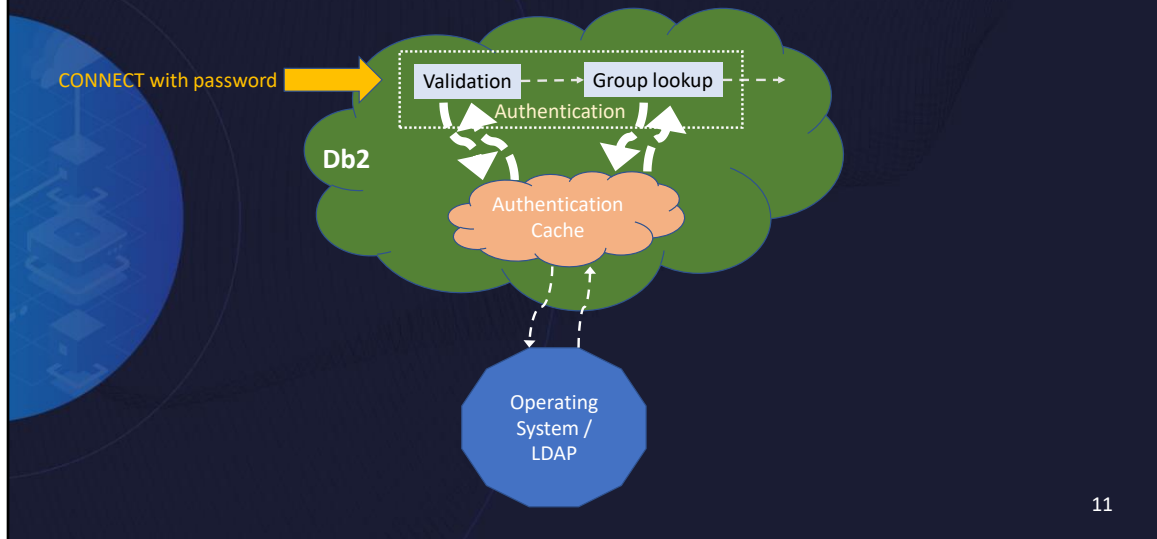
Without the authentication cache



10

Validation and Group Lookup both have to go back to the OS

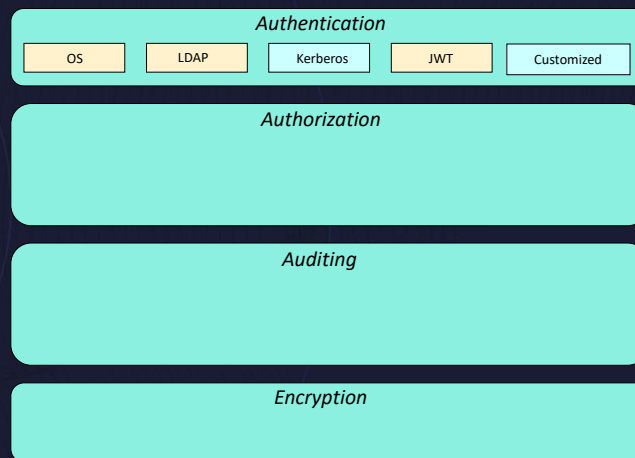
With the authentication cache



11

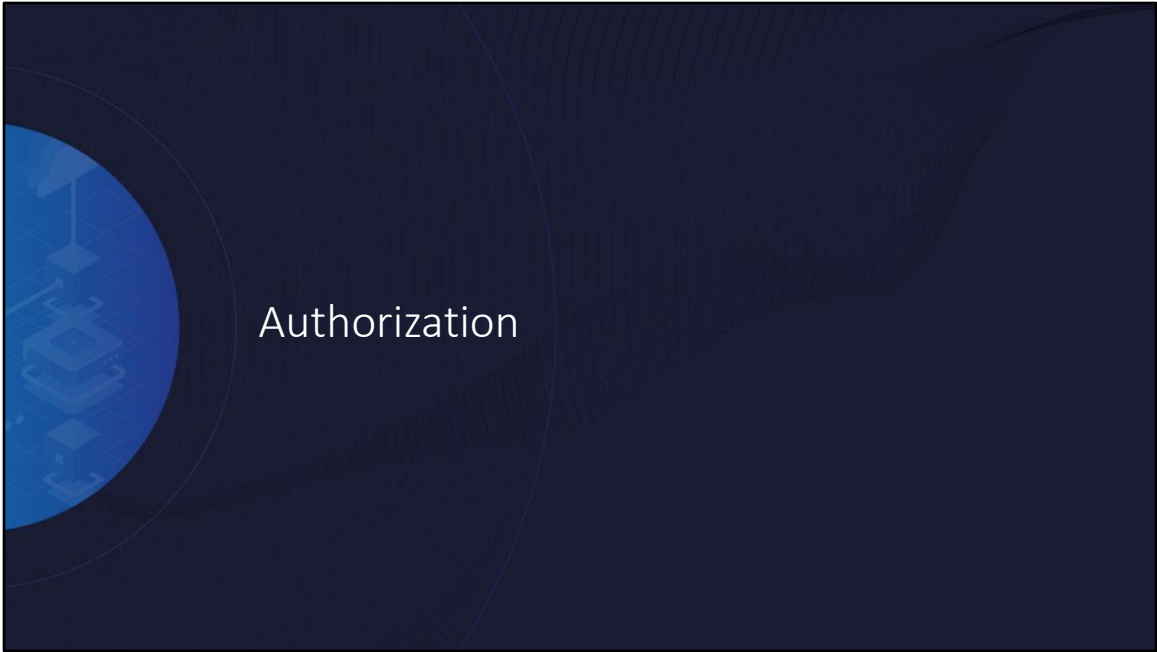
Look in authentication cache first
If not found, query the OS

Database security landscape



12

As a reminder, the “peach” coloured boxes indicate something new or enhanced in Db2 11.5.



Authorization

Authorities & privileges: What are they?

- Authorities and privileges are explicitly declared permissions within Db2 used to allow users to perform specific actions
 - An action is authorized based on the collection of authorities and privileges held, directly or indirectly, by an authorization ID
- Authorities represent a predefined collection of Db2 permissions within a specific domain
- Privileges represent Db2 permissions on a specific database object
 - Privilege is on a specific instance of an object (not a specific type of object)

Primary Db2 authorities

Instance

- ❖ SYSADM
- ❖ SYSCTRL
 - ❖ SYSMAINT
 - ❖ SYSMON

Database

- ❖ DBADM
 - SQLADM
 - EXPLAIN
 - WLMADM
- ❖ SECADM
 - ACCESSCTRL
- ❖ DATAACCESS

Schema (11.5.5.0 for Db2)

- ❖ SCHEMAADM
 - LOAD
- ❖ ACCESSCTRL
- ❖ DATAACCESS

Caution: DBADM GRANT statements

- By default, GRANT DBADM also implicitly grants DATAACCESS and ACCESSCTRL authorities
- Only do this if they really need it!

```
>>-GRANT----->
|
|-----|
V-----|
>-----ACCESSCTRL----->
+-----+
+ BINDADD-----+
+ CONNECT-----+
+ CREATETAB-----+
+ CREATE_EXTERNAL_ROUTINE-----+
+ CREATE_NOT_FENCED_ROUTINE-----+
+ CREATE_SECURE_OBJECT-----+
+ DATAACCESS-----+
|-----+-----+-----+-----+-----+-----+-----+-----+-----+
|-----WITH DATAACCESS-----|-----WITH ACCESSCTRL-----|
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|-----WITHOUT DATAACCESS-----|-----WITHOUT ACCESSCTRL-----|
+-----+
+ EXPLAIN-----+
+ IMPLICIT_SCHEMA-----+
+ LOAD-----+
+ QUIESCE_CONNECT-----+
+ SECADM-----+
+ SQLADM-----+
+ WLMADM-----+
|
```

16

Although DATAACCESS and ACCESSCTRL are intended to be independent “peer” authorities for DBADM, historically they have not been. DBADM used to be everything... When we separated out the major database authorities in DB2 9.7 (e.g. SYSADM no longer held DBADM, DBADM and SECADM were co-owners of database, DBADM authority did not mean DATAACCESS and ACCESSCTRL), we decided to minimize the shock to operations on existing databases by continuing to grant these 2 authorities with the DBADM authority by default. In other words, you have to opt=out of the traditional behaviour.

Additional schema level privileges (11.5.5.0 for Db2)

SELECTIN	Gives the ability to retrieve rows from all existing and <i>future</i> tables or views defined in the schema
INSERTIN	Gives the ability to insert rows and to run the IMPORT utility on all existing and <i>future</i> tables or views defined in the schema
UPDATEIN	Gives the ability to use the UPDATE statement on all existing and <i>future</i> tables or updatable views defined in the schema
DELETEIN	Gives the ability to delete rows from all existing and <i>future</i> tables or updatable views defined in the schema
EXECUTEIN	Gives the ability run all existing and <i>future</i> user-defined functions, methods, procedures, packages, or modules defined in the schema

17

These new privileges are in addition to the existing CREATEIN, ALTERIN, and DROPIN schema privileges.

Authorities & privileges: Who can hold them?

- Authorities and privileges can be associated to a specific Db2 authorization ID
- A Db2 authorization ID consists of:
 - Authorization ID type
 - Unique authorization ID value (128-byte limit)
- Authorization ID types:
 - Individual user ('U')
 - Group ('G')
 - Role ('R')
 - PUBLIC ('P')
 - Represents all authorization IDs in the "universe"

How the different authorization ID types interact

- Authorization processing considers both a **primary** authorization ID and **secondary** authorization IDs associated with the primary ID
 - Specific IDs considered are determined by the context
- **Primary** authorization ID
 - Used to record “who” performed the action
 - Represents an individual user ID ('U')
- **Secondary** authorization ID
 - One or more authorization IDs associated with the primary authorization ID
 - Used to supplement the primary ID's privileges **where allowed by Db2**
 - Represents groups, database roles, and/or PUBLIC

Common Db2 authorization ID terminology

- **SYSTEM AUTHORIZATION ID**
 - Primary authorization ID (and associated secondary authorization IDs) used to establish the current session and is checked for CONNECT privilege
- **SESSION AUTHORIZATION ID**
 - Primary authorization ID (and associated secondary authorization IDs) used for any session authorization checking after CONNECT processing.
 - The value of the SESSION AUTHORIZATION ID is controlled by the authentication logic but is typically the same value as the SYSTEM AUTHORIZATION ID
- **STATEMENT AUTHORIZATION ID**
 - Primary authorization ID (and associated secondary authorization IDs) used for any authorization requirements of an SQL statement
 - Also used to determine object ownership (for DDL where appropriate).
 - Can vary depending on the type of SQL statement and the context in which the statement is issued
 - Different sources for dynamic SQL versus static SQL and different options available for routine or non-routine context

Authorities & privileges: How do you get them?

- Authorities and privileges can be acquired permanently through **explicit** and **implicit** mechanisms
 - GRANT and REVOKE SQL statements are the **explicit** mechanisms
 - Can be used for an authorization ID representing user, group, or role
 - Object creation and removal are examples of an **implicit** mechanism
 - E.g., Object owner granted some permissions by Db2 as a result of the creation
 - Note that other implicit mechanisms exist
- Authorities and privileges can also be acquired **temporarily** through different execution contexts
 - E.g., You have access to them only while you are in that execution context
 - Examples of different execution contexts include static SQL, views, routines, and trusted contexts

Temporary access to authorities and privileges

- Inheritance

- A privilege on a view implicitly gives you the same access to the objects in the view definition when using the view
 - E.g., Inserting into a view, inserts into a table in the view definition
- Execute privilege on a package gives you the right to execute any static SQL in that package and inherit the package owner's privileges through that SQL
 - E.g., Executing a package with static DELETE statement, lets you delete from that object

- Alternate authorization models

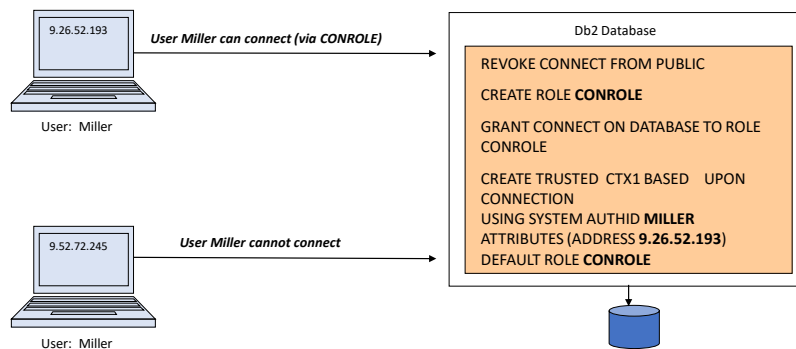
- Package **DYNAMICRULES** bind option lets you specify which authorization ID is used for embedded dynamic SQL issued by that application
 - Possible options include able to have the primary authorization ID be the Package Definer or Executor, the (associated) Routine Definer or Invoker

- Trusted context

A trusted context is...

- A declaration of a “trust relationship” between the database and an external application based on a set of explicit of trust attributes:
 - System authorization ID
 - IP address
 - Level of communication security
- A connection that matches the trust attributes for a defined trusted context is called a trusted connection. There are 2 types:
 - An **implicit** trusted connection
 - An **explicit** trusted connection
- An implicit trusted connection allows a user to inherit a role that is not available to them outside the scope of that trusted connection
 - The session authorization ID of the connection is given “temporary” membership to a role declared in the trusted context definition

Implicit trusted connection: Role inheritance



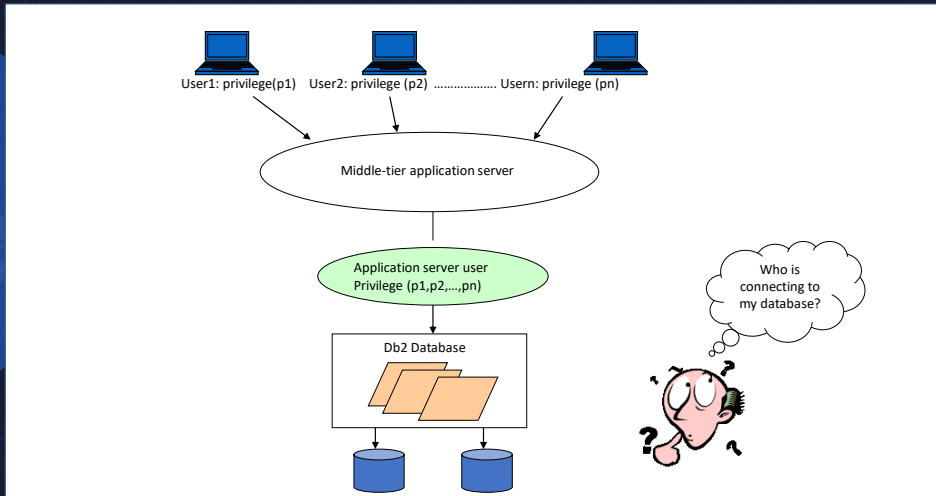
24

In this example, we have removed CONNECT authority from PUBLIC so that no one can connect to the database by default. We then create a role and give it CONNECT authority on the database. We then define a trusted context saying that any connection established using the MILLER authorization ID from the IP address 9.26.52.193 will be implicitly trusted and will therefore gain access to the role... and thus will be able to connect to the database.

Explicit trusted connections

- An explicit trusted connection allows a trusted application server to switch, or assert, the current end-user ID on the existing connection in an efficient manner
- An application server establishes the original connection with an explicit request for trust and, once established, it can then issue requests to the database server to change the session authorization ID for any new unit of work
 - The ID used to do the initial connect request for the application server only needs `CONNECT` privilege

Who is doing what ?



26

In environments where application servers have to deal with many end-user clients, it is not easy/practical to have the application server establish a separate connection to the database for each new user

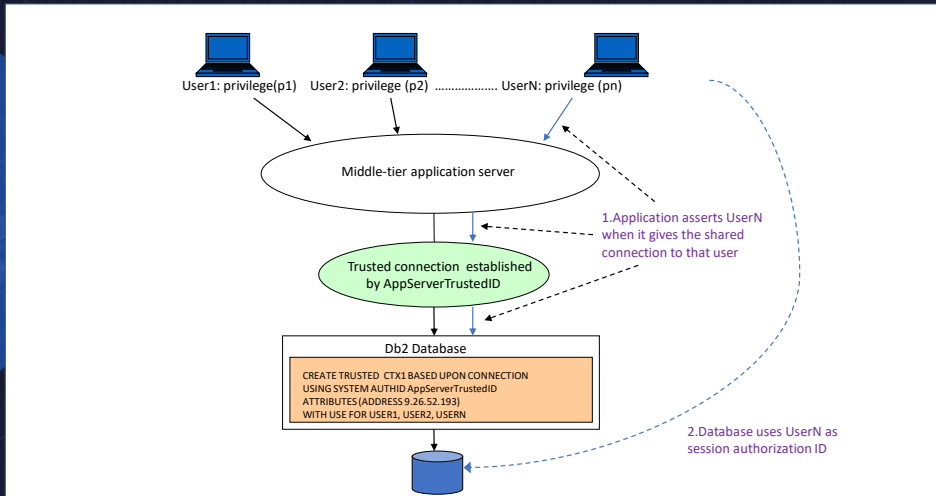
- Overhead of creating a physical connection
- Overhead of user authentication

In some cases, this is not even an option as the application server may not have the end user credentials to make a connection to the database. To get around this, customers often end up using a common, shared user ID to connect to the database and to pass along the end-user database requests. The common user ID becomes a "super ID" with a union of all the privileges needed by the applications serviced by the application server.

This makes database auditing and authorization functions effectively useless

- You cannot tell the auditor who did what
- You cannot prevent one user from doing what another user can

Identity assertion model



27

With a trusted context:

1. User connects to the application
2. Application creates a trusted connection with the database
3. Application switches the user for the trusted connection to UserN
4. Actions can be done on the database as UserN

Audit the correct user

Authorize the correct users for different functionality

Advanced authorization controls

- Sometimes there are requirements to implement authorization controls that operate within a table object itself at the row, column, or cell level
 - Such controls are referred to as “fine grained access controls” (FGAC)
- FGAC supplement traditional authorization controls and allow security administrators to control the results sets seen by different people even when they run the same SQL statement
 - Your privileges do not control what data you can see with FGAC
- Db2 offers two variations of FGAC
 - Label-Based Access Control (LBAC)
 - Row and Column Access Control (RCAC)

28

Label-Based Access Control (LBAC)

- LBAC is an implementation of a Mandatory Access Control (MAC) system
 - Both the users and the data itself are explicitly assigned a security label value
 - The intersection between the user security label and the data security label determines what rows and columns can be seen by each user
 - Based on a set of pre-defined rules on how different security labels interact
- A key prerequisite for LBAC is a clear definition of security labels and assignments
 - Change is very difficult to propagate as labels are part of the data itself
- Primary use case is in traditional military and intelligence domains
 - Is used in some commercial environments



29

Row and Column Access Control (RCAC)

- RCAC is based on the use of simple, flexible SQL to express customer supplied rules
 - Unlike LBAC, no need to define and assign security labels and no impact on the underlying data
- RCAC consists of two components:
 - Row permissions
 - An SQL search condition that describes what set of rows can be accessed
 - Column masks
 - An SQL CASE expression that describes what column values are permitted to be seen and under what conditions

Row permission example

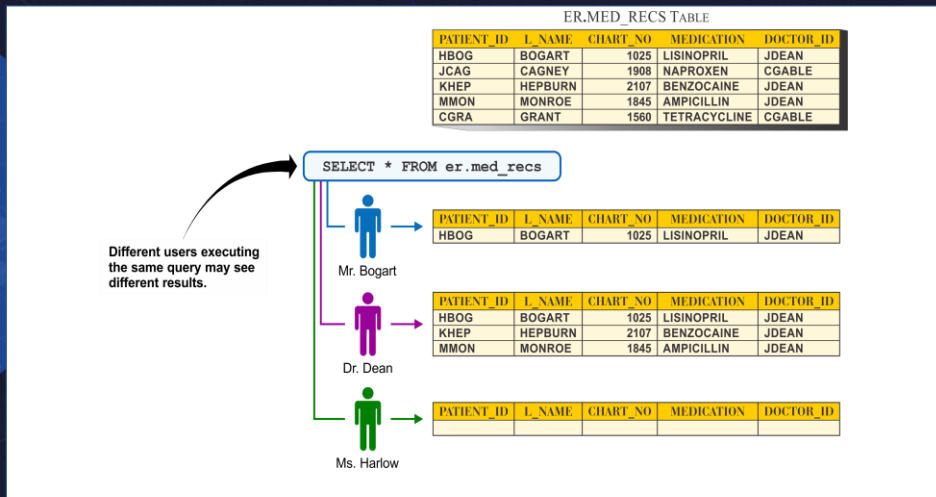
```
CREATE PERMISSION row_access ON er.med_recs
FOR ROWS WHERE
  (VERIFY_ROLE_FOR_USER(SESSION_USER, 'PATIENT') = 1
   AND er.med_recs.patient_id = SESSION_USER)
OR
  (VERIFY_ROLE_FOR_USER(SESSION_USER, 'DOCTOR') = 1
   AND er.med_recs.doctor_id = SESSION_USER)
ENFORCED FOR ALL ACCESS
ENABLE;

ALTER TABLE er.med_recs ACTIVATE ROW ACCESS CONTROL;
```

31

This example row permission is set to allow access to the current row if the person accessing it is a patient (determined by PATIENT role membership) looking at their own record (determined by session authorization ID) or a doctor looking at one of their own patients.

Row permissions in action



32

In the example shown on this slide, three different users can execute the same query, however they will see different results because of the row permission that was defined earlier.

Column mask example

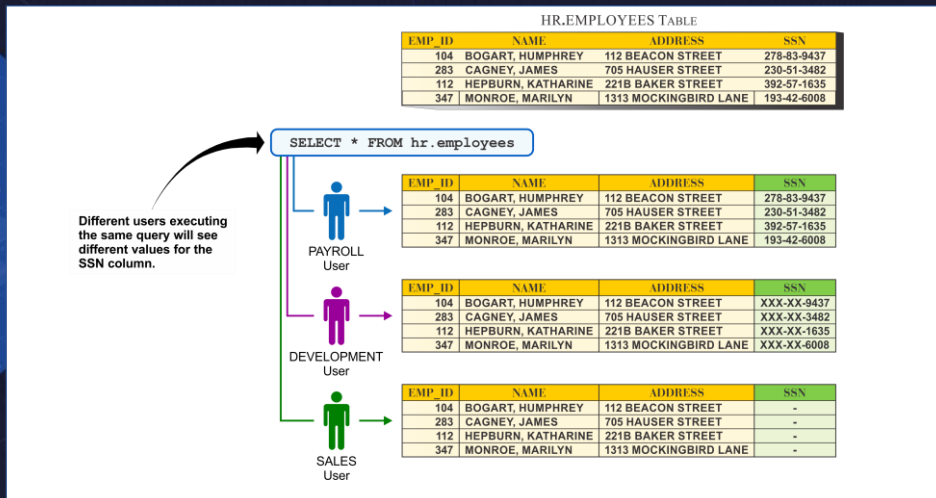
```
CREATE MASK ssn_mask ON hr.employees
FOR COLUMN ssn RETURN
CASE
  WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, 'PAYROLL') = 1
  THEN ssn
  WHEN VERIFY_ROLE_FOR_USER(SESSION_USER, 'DEVELOPMENT') = 1
  THEN 'XXX-XX-' || SUBSTR(ssn, 8, 4)
  ELSE NULL
END
ENABLE;

ALTER TABLE hr.employees ACTIVATE COLUMN ACCESS CONTROL;
```

33

An example column mask where if you belong the PAYROLL role, you can see the entire social security number and if you belong to the development team, you can see a masked version of the number. If you are neither, you see a NULL result.

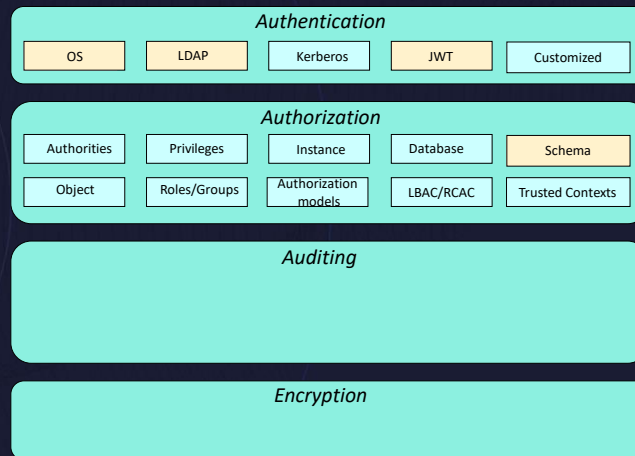
Column masks at work

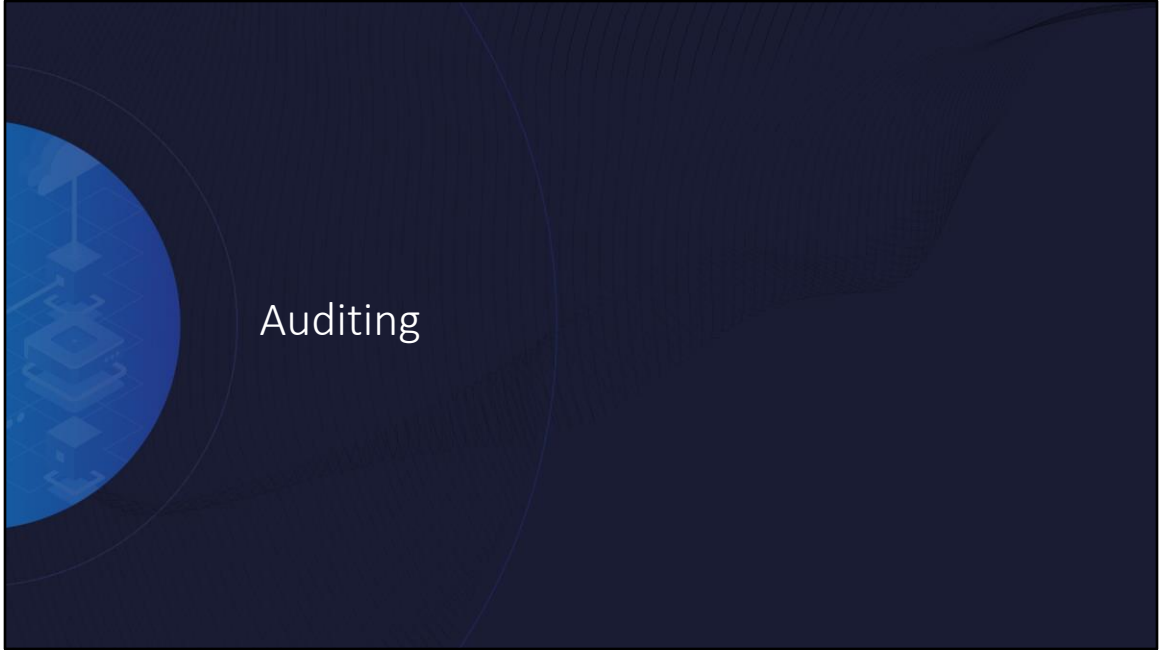


34

In the example shown on this slide, three different users can execute the same query, however they will see different results because of the row permission that was defined earlier.

Database security landscape





Auditing for Db2

- Two primary options offered with Db2:
 - Integration with an external product (IBM Guardium)
 - Db2 Audit facility
- Integration with IBM Guardium
 - Done using DRDA communication buffer exit on both send and receive
 - Offers programmatic ability to terminate a connection if desired
- Db2 audit facility
 - Internal audit capability for a pre-defined set of events which provide insight into who did what, when, and where

37

Of course, you can also build your own audit capability using triggers or application logic.

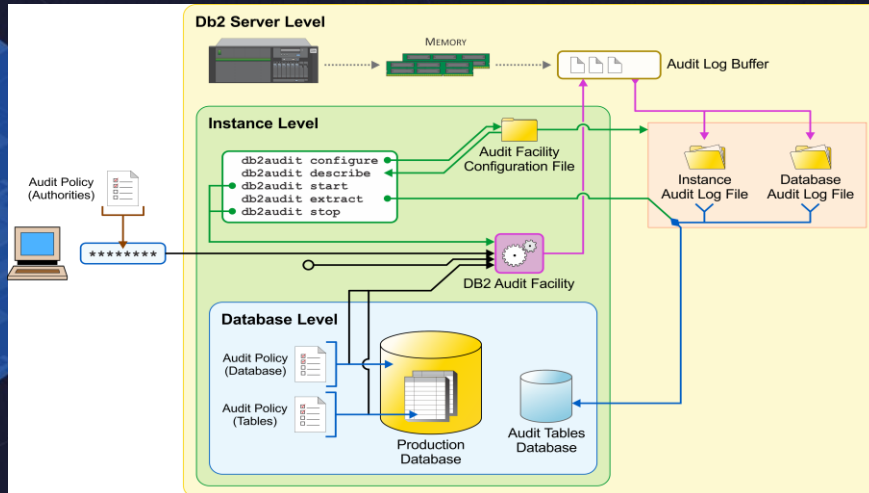
Db2 audit facility

- Audit can be configured at both instance level and within each database
 - Separate audit log for instance and each database
- Configuration can specify desire to audit one or more of the defined event categories:
 - **AUDIT**: Change in audit settings or audit log access
 - **CHECKING**: Authorization checks
 - **OBJMAINT**: Objects created or dropped (some but not all alterations)
 - **SECMAINT**: Changes to security controls
 - **SYSADMIN**: Use of SYSADM, SYSMAINT, or SYSCTRL authority
 - **VALIDATE**: Authentication or access of system security information
 - **CONTEXT**: Shows contextual information for a database operation
 - **EXECUTE**: Execution of SQL statements

Granularity of database auditing

- Database audit is defined using audit policies which are then associated with specific objects using the AUDIT statement
- Audit policies can be associated with different database objects to control what is audited
 - The database itself
 - Tables
 - Authorities such as SYSADM, DBADM, and SECADM
 - Users and groups
 - Roles
 - Trusted Connections
- This granularity allows a narrowed focus to be applied on for audit
 - Can result in significant reductions in the amount of audit data

The Audit Facility – Illustrated



40

A high-level overview of how the Db2 Audit Facility works is depicted on this slide.

Instance level

- `db2audit configure` writes to the audit facility configuration files – configures instance level audit
- `Db2audit describe` shows the configuration
- `Db2audit start/stop` controls the instance level audit
- Logged into instance audit file

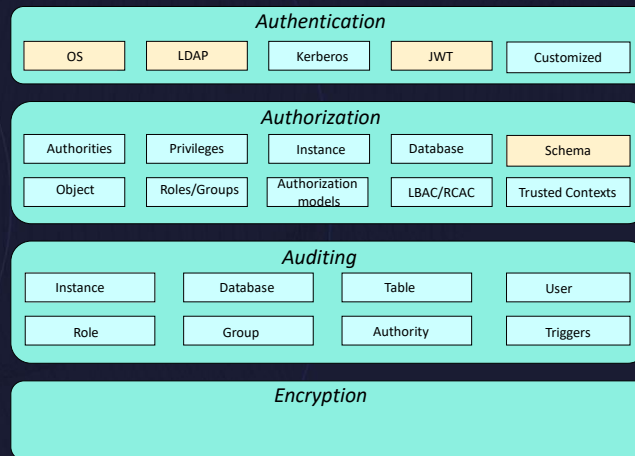
Database level

- Audit policies define what get audited in the database
- Logged into database audit file

File extraction

- File gets archived
- Extract into either table or somewhere else

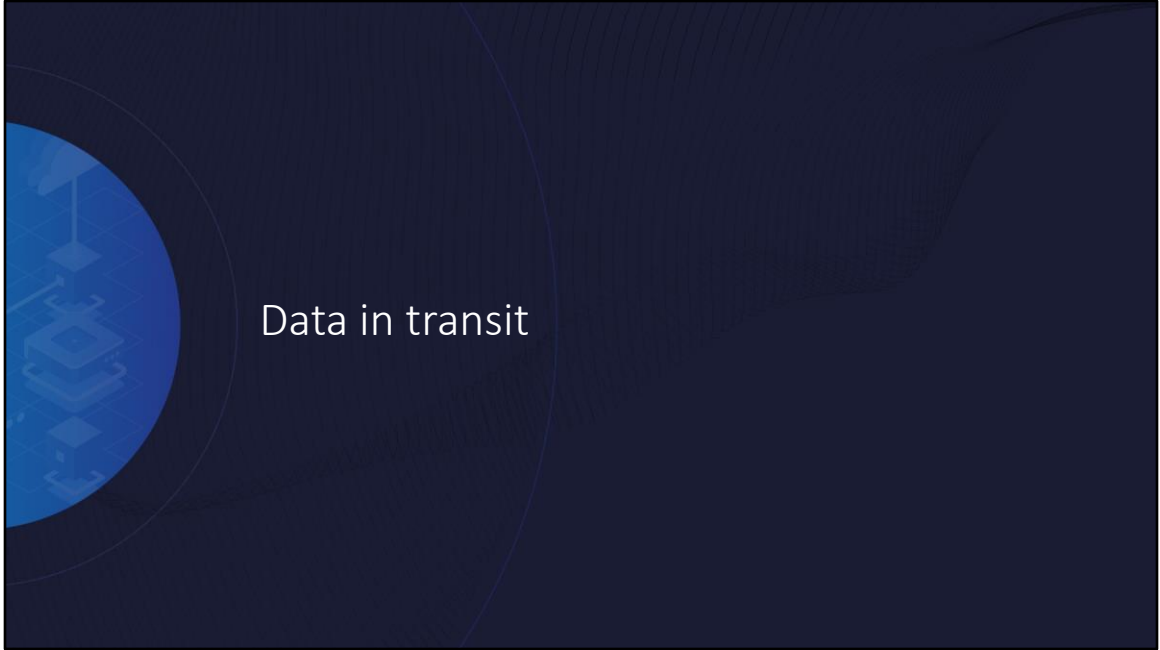
Database security landscape





Two focus areas

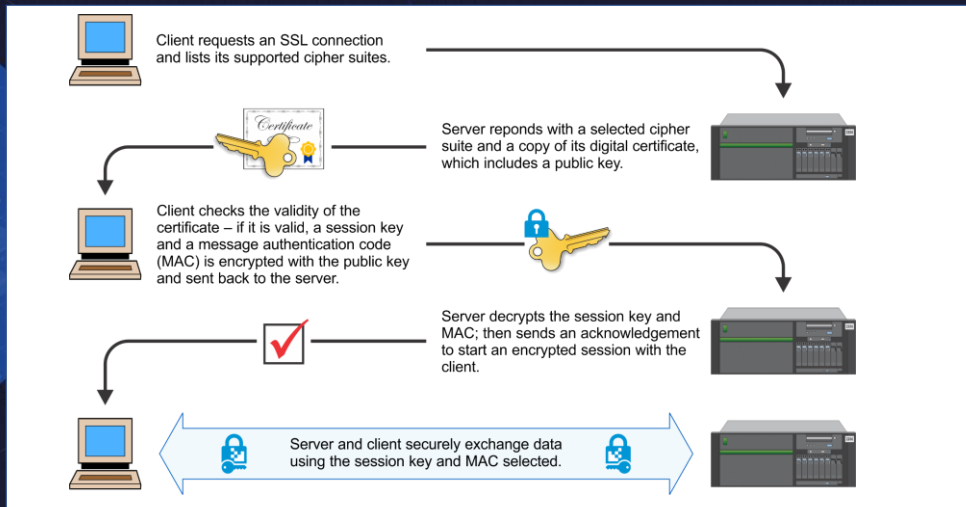
- Guarding communications → Data in transit
- Guarding database storage → Data at rest



Guarding communications

- Where is it relevant?
 - Between client and server
 - Between HADR primary and standby(s)
 - Between Db2 and external products or services (e.g., security products, remote storage repositories)
- How is it done?
 - Communications are protected by encrypting the data being transmitted using a mechanism generically referred to as SSL (especially in Db2 documentation)
 - The current specific industry standard recommendation is TLS 1.3 (Db2 supports this as of 11.5.8) and TLS 1.2
 - SSL = Secure Sockets Layer
 - TLS = Transport Layer Security

"SSL handshake"



46

How SSL Works With Db2

A client and server establish a secure SSL connection by performing an "**SSL handshake**". During an SSL handshake, a public-key algorithm, usually RSA, is used to securely exchange digital signatures and encryption keys between a client and a server. This identity and key information is used to establish a secure connection for the session between the client and the server. After the secure session is established, data transmission between the client and server is encrypted using a symmetric algorithm, such as AES.

The client and server perform the following steps during the SSL handshake:

1. The client requests an SSL connection and lists its supported cipher suites.
2. The server responds with a selected cipher suite.
3. The server sends its digital certificate to the client.
4. The client verifies the validity of the server certificate, for authentication purposes. It can do this by checking with the trusted certificate authority that issued the server certificate or by checking in its own key database.
5. The client and server securely negotiate a session key and a message

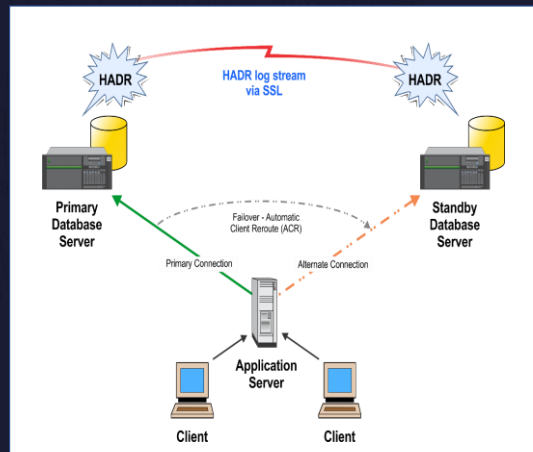
authentication code (MAC).

6. The client and server securely exchange information using the key and MAC selected.

Note: The Db2 database system does not support the (optional) authentication of the client during the SSL handshake.

SSL between HADR Primary and Standby servers

- Provides integrated protection of sensitive data in the log stream
- Enabled via the HADR_SSL_LABEL database configuration parameter
- Supports all HADR synchronization modes
- Supports multiple standbys



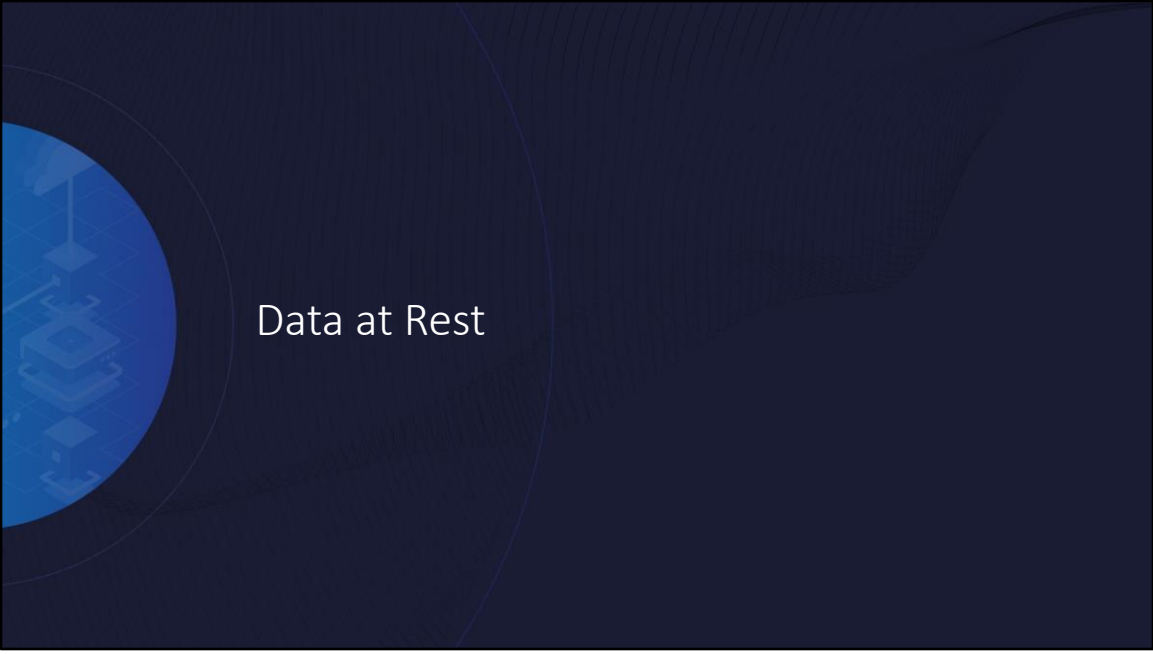
47

When implementing HADR in a Db2 11.1.1.1 environment, you can use SSL technology to secure TCP/IP communications and encrypt the data that transferred via log streams between the primary and the standby server—but only for non-DB2 pureScale deployments on x86 Linux platforms.

Refer to the section “**Configuring SSL for the communication between primary and standby HADR servers**” in the v11.1 DB2 LUW Knowledge Center for more information on how to use this new security capability.

Remember SSL certificates expire!

- Certificates have an expiration date associated with them
 - Once that date (and time) has passed, SSL negotiation will fail
- You will need to update the certificates being used by Db2 in the affected area(s)
 - Client/server
 - HADR
 - Keystore
- In some of these, a restart of DB2 will be required
 - <https://www.ibm.com/support/pages/do-we-need-restartrecycle-db2-after-revisingrenewing-ssl-certificate>
 - As of Db2 11.5.2.0, the SSL_SVR_LABEL database manager parameter can now be updated dynamically (client/server SSL)



Data at Rest

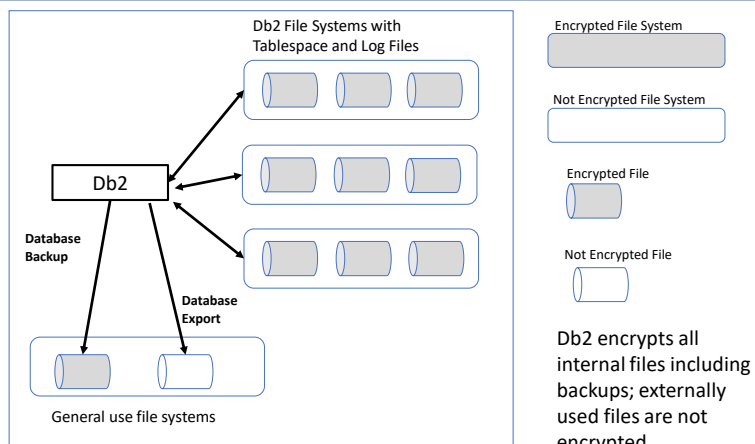
Guarding (on-disk) database storage

- Where is it relevant?
 - Files containing user data such as database files, backup images, and transaction logs
- How is it done?
 - File system encryption such as with Guardium Data Encryption (GDE)
 - Db2 native encryption

Db2 Encryption

- Db2 Native Encryption (naturally) encrypts Db2 databases only
 - Encrypts your data as it is written to disk.
 - The encryption is implemented within Db2 itself.
- Db2 encrypts all internal files including backups but externally used files are not encrypted

Db2 Encryption Only



52

Database file systems with tablespaces and logs, backups all encrypted
Externally used files like exported files are not encrypted

Db2 & GDE review

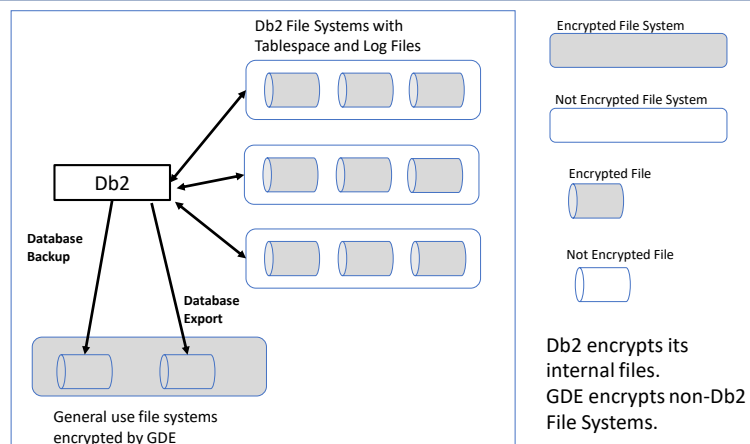
- Similarities

- Both are FIPS 140-2 certified cryptographic libraries
- No difference in terms of encryption strength

- Differences

- Db2 automatically/transparently handles encryption across all members
- Db2 works on all Db2 configurations
- Db2 encryption travels with the object (i.e., the object itself is encrypted)
- GDE allows for encryption of other databases besides Db2
- GDE applies to all files written to encrypted file system regardless

Db2 + GDE Encryption



54

Db2 still encrypts its own files and backups
Exported files now encrypted by Guardium

Db2 native encryption

- Db2 Native Encryption is built into Db2 to protect data when it is at rest
- Available in all Db2 offerings free of charge
- Automatically detects and uses CPU hardware acceleration when available
 - Intel AES-NI hardware acceleration
 - Power8 in-core support for the Advanced Encryption Standard (AES)

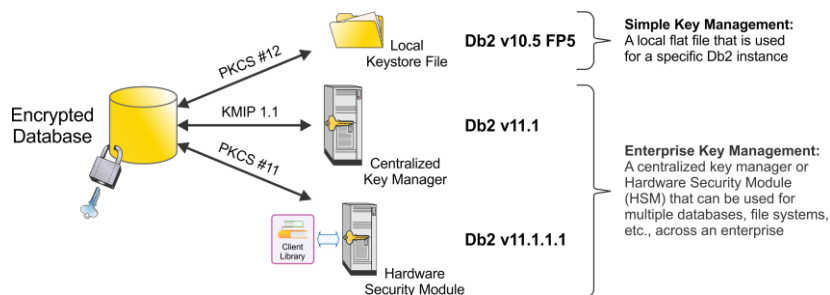
Highlights of Db2 Native Encryption

- Easy to deploy and works on all Db2 platforms
 - Transparent to applications!
 - No system administrator needed!
- Industrial strength
 - FIPS 140-2 certified encryption libraries
 - NIST compliant use of cryptography (e.g., NIST SP 800-131)
- Secure and transparent key management
- Encrypts all critical data

Basics of key management

- Db2 Native Encryption uses an industry standard 2-tier model
 - Actual data is encrypted with a Data Encryption Key (DEK)
 - DEK is encrypted with a Master Key (MK)
- DEK is managed within the database while the MK is stored externally in a keystore

Keystore options



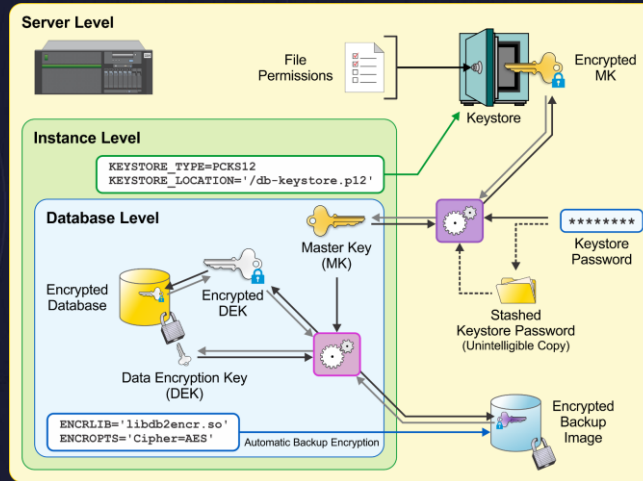
58

Db2 Native Encryption uses an industry standard, 2-tier model for key management

- Data is encrypted with a Data Encryption Key (DEK)
- The DEK is encrypted with a Master Key (MK) and the encrypted DEK is stored within the database (or in backup images)

The Master Key (MK) is managed externally and can be stored in a PKCS #12 compliant local keystore, which is created using the IBM Global Security Kit (GSKit). Db2 V11.1 added support for KMIP 1.1 compliant centralized key managers like IBM Security Key Lifecycle Manager (ISKLM) and Safenet KeySecure. And Db2 v11.1.1.1 offers direct support for Hardware Security Modules (HSMs) – integration with Gemalto (formerly Safenet) Luna SA HSM and Thales NShield Connect+ is provided.

Local PKCS#12 keystore



59

Db2 Native Encryption – Local Keystore

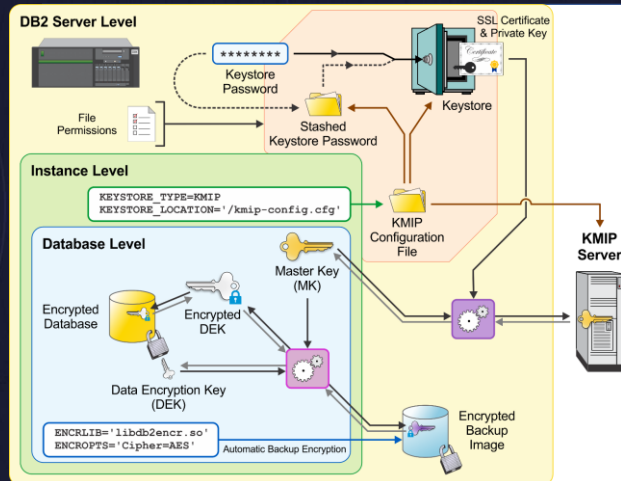
Db2 native encryption uses a standard two-tier model for key management—the Data Encryption Key (DEK) constitutes the first tier; a second key known as the Master Key (MK) constitutes the second. (This model is referred to as envelope encryption in the security industry.) The DEK is the actual key used to perform data encryption. The DEK is then encrypted with the MK and stored within the database (or backup image).

The MK is stored outside the database in a Public-Key Cryptography Standards (PKCS#12) compliant keystore. There are two security protection measures for your keystore. The first is file permissions. You need to make sure that only the Db2 instance owner has read/write access to the keystore. The second is encryption of the actual content of the keystore. You need to make sure you create your keystore with the password option. The content of the keystore (i.e., your master keys) is encrypted using a symmetric key derived from that password using a hashing algorithm. Without the password, the content of the keystore cannot be decrypted.

You have the option to stash or not stash the password. Stashing the password is

good for secure production environments where you need your Db2 instance to be able to start without human intervention. You can also choose not to stash the password and provide it only as needed when starting your Db2 instance. This is possible through the new open keystore option of the **Db2start** command. Db2 native encryption also allows you to rotate your MK—you rotate your database MK by calling the **ADMIN_ROTATE_MASTER_KEY** procedure. This procedure decrypts the database DEK with the old MK and then re-encrypts it with the new MK. You have 2 options when calling the **ADMIN_ROTATE_MASTER_KEY** procedure: you can either provide a label for the desired new MK or use the default. When using the default, Db2 automatically generates a new MK and adds it to the keystore on your behalf. Then, it rotates the current database MK to this newly generated MK.

Centralized key manager



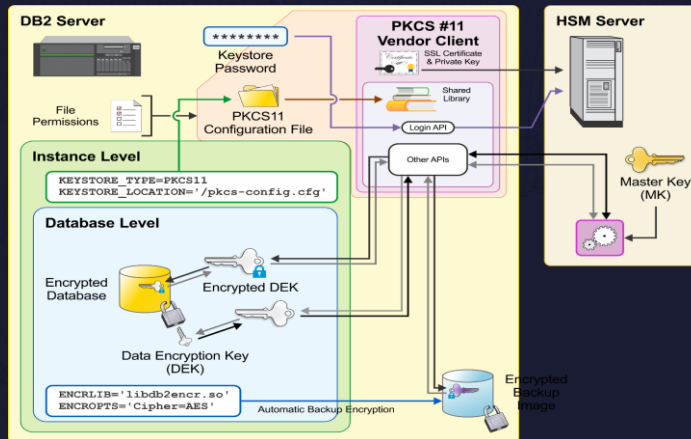
60

Db2 Native Encryption – Centralized Key Manager

Beginning with Db2 v11.1, native encryption can utilize an enterprise key management system, which is a dedicated server for centrally managing encrypting keys across the enterprise. Some of the more common enterprise key management systems include IBM Security Key Lifecycle Manager (ISKLM), Safenet, and KeySecure.

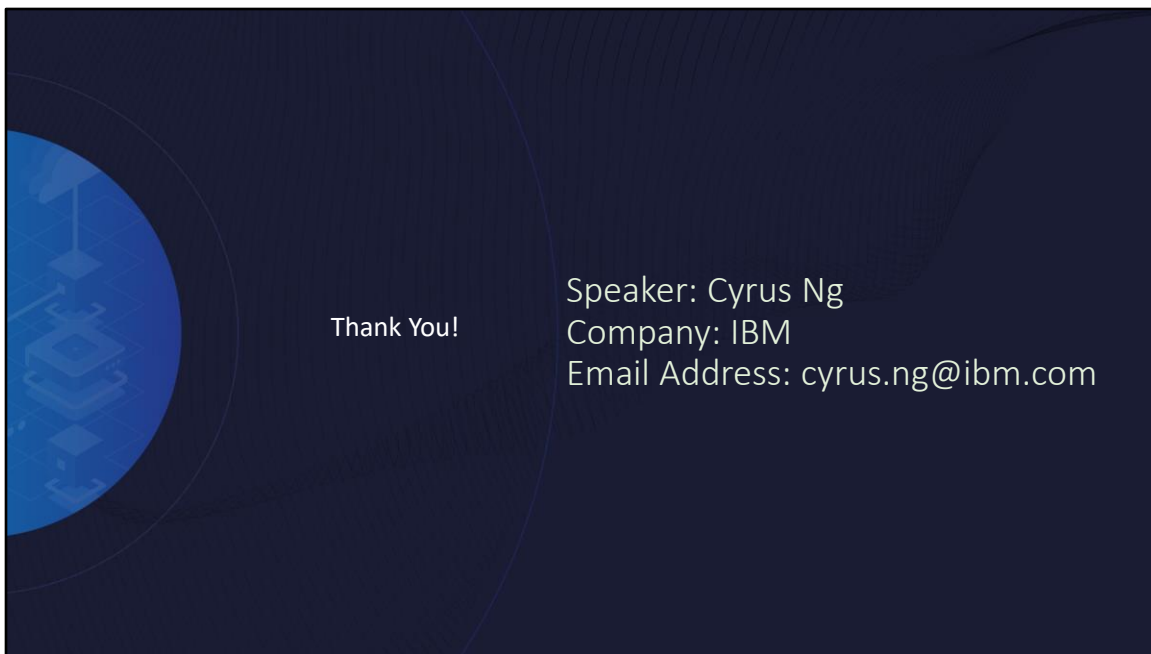
The Key Management Interoperability Protocol (**KMIP**) is a communication protocol that defines message formats for the manipulation of cryptographic keys on a key management server.

Hardware Security Module (HSM)



Considering encryption?

- Encryption is not something to rush into as it has implications for availability, operations, and performance!
- Availability:
 - <https://www.ibm.com/docs/en/db2/11.5?topic=considerations-keystore-availability-recoverability>
 - Keystore availability issues now become data availability issues
- Operations:
 - <https://www.ibm.com/docs/en/db2/11.5?topic=considerations-impact-encryption-database-operations>
- Performance:
 - <https://www.ibm.com/docs/en/db2/11.5?topic=considerations-impact-encryption-performance>
 - CPU hardware acceleration is critical
 - You should plan on completely re-tuning a newly encrypted system



Mitchell has been a member of the Db2 security development team since 2016, where he has worked on all aspects of security within Db2, including authentication, authorization, auditing and encryption.