Securing Data with Apache Derby

Dan Debrunner
djd@debrunners.com

Agenda

- **Security Environment**
- **Database Encryption**
- **Database Authentication**
- **Database Authorization**
- **Database Properties**
- **Java 2 Security Manager**
- **Wrap-up**

Enterprise Database Location

- **Locked in machine room**
- **Physical access using card-key with recording**
- **Video cameras inside & outside machine room**
- **Armed Guard**
- **Backups under controlled access**

Enterprise Database Security

- **Operating System authentication**
  - No general login to database server
- **Operating System authorization**
  - no general read or write access to db devices or files
- **Database authentication**
  - Trusted users only
- **Database authorization**
  - Grant/revoke to limit operations for each user or group
- **Application authorization/authentication**

Security Warning

- **Application and/or database level security can not solely protect data**
- **Therefore reliance on them can lead to a false sense of security.**
- **E.g. using GRANT/REVOKE to disallow update access to the salary table to Fred does not provide any security if Fred can modify the data on disk directly, to award himself a 7 figure salary**
- **This is true for any database system**

Derby Security Environment

- **A typical Derby database is not maintained on disks in a server machine in a locked room with armed guards or video surveillance**
- **A typical Derby database is on a laptop, desktop, kiosk, machine in a closet, machine in a cell-tower, etc.**
- **This means the data can be easily accessed without going through the application or database engine**
  - Steal the disks and use a binary editor

Derby Encryption

Derby Database Encryption

- **Complete encryption of the on-disk data**
  - Data files for indexes and tables
  - Transaction Log File
  - Temporary files (for ORDER BY etc.)
- **Includes application data and system data**
  - Table data
  - All system catalog/metadata information

What's Not Encrypted

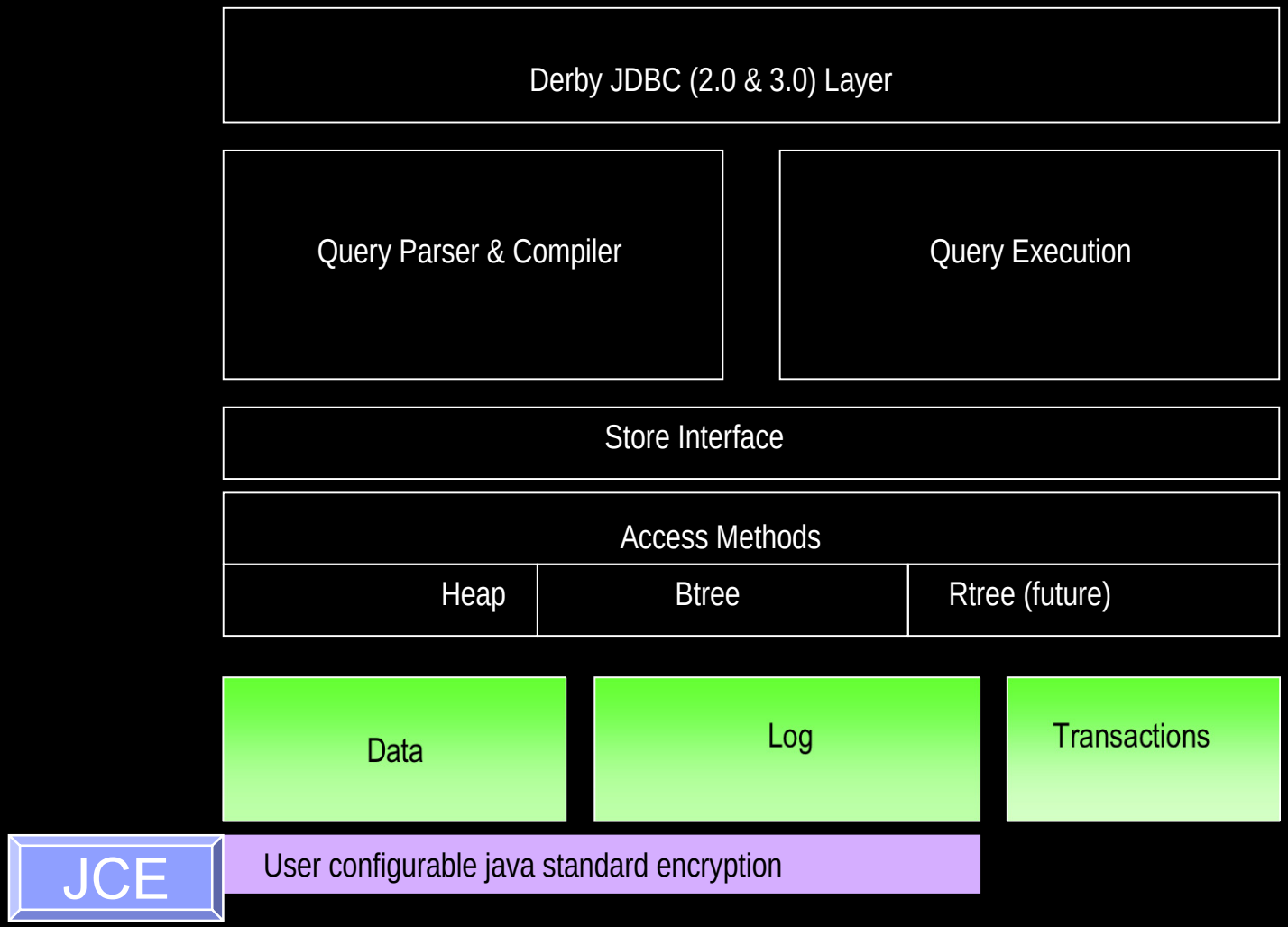
- **Data in-memory**
  - Page cache contents
  - ResultSets
- **service.properties file**
  - Only contains minimal information to boot database but can contain some encryption related information
- **Jar files stored in the database through sqlj.install_jar**
- **derby.log – error log**

I/O Based Encryption

- **Data encrypted just before write call to disk**
- **Data decrypted immediately after read from disk**

- **Majority of system unaware of encryption**

Derby Architecture



Java Cryptography Extension (JCE)

- **Optional package in J2SE 1.3**
- **Integrated into J2SE 1.4**
- **Pluggable framework architecture that enables only qualified providers to be plugged in.**
- **Sun & IBM's JVMs come with a provider**
- **Can utilize a third party provider if required**
  - Sun site lists five provider implementations
- **http://java.sun.com/jce**

JCE Algorithm

- **Provider implements encryption algorithms, loadable by name in Java code**
- **E.g. SunJCE provider**
  - DES, DESede, AES (with Java 2 SDK, v 1.4.2), Blowfish
- **Encryption strength settable by policy files, to allow (US) exportable JCE providers and non-exportable providers with larger key size**

Database Creation

- **Database configured for encryption at create database time**
- **Remains encrypted with same key forever**
- **Set via encryption attributes on the JDBC connection request, at least dataEncryption=true**
- **Two modes**
  - database key storage
  - external key storage

Default Provider Selection

- **J2SE 1.4 or higher, the JRE's provider is the default**
- **Sun J2SE 1.3 the default is *com.sun.crypto.provider.SunJCE***
- **IBM J2SE 1.3 the default is *com.ibm.crypto.provider***
- **Other J2SE 1.3 a provider must be specified**

Alternate Provider Selection

- **Connection attribute** encryptionProvider
- **Set to class name of provider, e.g.** org.bouncycastle.jce.provider.BouncyCastleProvider
- **Application needs to ensure provider's jar is in classpath**
- **Requires Java 2 Security manager permission**

Encryption Algorithm

- **Defaults to DES**
- **Settable by connection attribute *encryptionAlgorithm***
- **Format**
  - *algorithmName/feedbackMode/padding*

algorithmName/feedbackMode/padding

- **JCE naming convention**
- **Any encryption algorithm that fulfills the following requirements:**
  - it is symmetric
  - it is a block cipher
  - it uses the *NoPadding* padding scheme
  - its secret key can be represented as an arbitrary byte array
  - it requires exactly one initialization parameter, an initialization vector of type *javax.crypto.spec.IvParameterSpec*
  - it can use *javax.crypto.spec.SecretKeySpec* to represent its key
  - FeebackMode one of CBC, CFB, ECB or OCB

Key Strength

- **DES 56 bits**
- **DESede 168 bits**
- **Other algorithms**
  - Default to 128 bits
  - Settable by connection attribute (in bits) encryptionKeyLength

Database Key Store

- **Encryption key generated by Derby**
- **Encryption key stored in service.properties file**
- **Key encrypted by "boot password" using same algorithm and strength as selected for the database**
- **Boot password set using connection attribute bootPassword**
- **Length of boot password should at least match key length**
  - 56 bits -> 8 characters
  - 168 bits -> 24 characters
  - 128 bits -> 16 characters
- **Minimum of 8 characters accepted by Derby**

Creation using DES Encryption

- jdbc:derby:db1;create=true;**dataEncryption=true;bootPassword=BU72x0tON65**

- ```
  Properties p = new Properties();
  p.put('create', 'true');
  p.put('dataEncryption', 'true');
  p.put('bootPassword', '8win713shi331l');
  Connection conn =
    DriverManager.
      getConnection('jdbc:derby:db2', p);
  ```
  ```

Creation using 128bit Blowfish Encryption

- jdbc:derby:db3;create=true;dataEncryption=true;
  bootPassword=Sut3TON_cu6m_Duc67KMAntON;
  encryptionAlgorithm=Blowfish/CBC/NoPadding

- ```
  Properties p = new Properties();
  p.put('create', 'true');
  p.put('dataEncryption', 'true');
  p.put('bootPassword',
  'C4oton_41in_THE_eLMs891');
  p.put('encryptionAlgorithm',
  'Blowfish/CBC/NoPadding');
  Connection conn =
    DriverManager.
      getConnection('jdbc:derby:db4', p);
  ```
  ```

Creation using BouncyCastle Provider & DESede

- jdbc:derby:db5;create=true;dataEncryption=true;
  bootPassword=5cha5pe4l_en_l2e_FrI4Th9;
  encryptionAlgorithm=DESede/CBC/NoPadding;
  **encryptionProvider=org.bouncycastle.jce.provider
  .BouncyCastleProvider**

Service.properties (cleartext) Information

- **db1 – DES**

```
dataEncryption=true
encryptionAlgorithm=DES/CBC/NoPadding
derby.encryptionBlockSize=8
encryptionKeyLength=56-8
encryptedBootPassword=b71c86f96f43a12a-30519
```

- **db3 – Blowfish**

```
dataEncryption=true
encryptionAlgorithm=Blowfish/CBC/NoPadding
derby.encryptionBlockSize=8
encryptionKeyLength=128-16
encryptedBootPassword=4b5bfedcde943607612ca5b201390540-1986
```

Potential Helpful Information for Crackers

- **Algorithm name**
- **Key size**
- **Encrypted key**
  - Cracking this opens the whole database
  - EncryptedBootPassword is a misnomer
- **Block size (?)**

External Key Store

- **Application provides encryption key at create database time**
- **Connection attribute encryptionKey=*encodedSecretKey***
- **Application relies on external secure key store, e.g. a smart card**

Key Encoding

- **The encodedSecretKey is a simple hexadecimal textual representation of the encoded form of the secret key returned by the method**
  - `byte[] java.security.Key.getEncoded()`
- **Format is simply each byte of the byte array represented by its two digit (character) hex value, i.e 00 to ff (case insensitive). The first byte in the array (offset 0) corresponds to the first two characters of the String**
- **Hard to believe, no standard Java api for this encoding**

Create with External Key

- **Generate key**
- **Encode key**
- **Create database**
  - dataEncryption=true
  - encryptionKey
  - [encryptionAlgorithm]
  - [encryptionProvider]
- **Key length is contained in key itself**

- **Notes has Java example class to create database**

Service.properties (cleartext) Information

- **db7 – AES – external key**

```
dataEncryption=true
derby.encryptionBlockSize=16
encryptionKeyLength=16
```

- **Better – but wrong**
- **Derby-42 bug – introduced when settable key length was added (with database key store)**
- **With external key storage no encryption information should appear in file**

Service.properties Information (post Derby-42)

- **db7 – AES – external key**
  `dataEncryption=true`

Booting an Encrypted Database

- **First connection needs to supply the correct boot password or encryption key**
- **Boot password verified using MD5 hash**
- **External key verified using encrypted random data (4k) and MD5 hash of unencrypted data**
  - verifykey.dat file in database folder
- **Keys verified to reduce chance of database corruption due to incorrect key**

Boot Using Boot Password

- **db1 (DES)**
  - `jdbc:derby:db1;bootPassword=BU72x0tON65`
- **db3 (BlowFish)**
  - `jdbc:derby:db3;`
    `bootPassword=Sut3TON_cu6m_Duc67KMAntON`
- **db5 (DESede with Bouncy Castle)**
- `jdbc:derby:db5;`
  `bootPassword=5cha5pe4l_en_l2e_FrI4Th9;`
  `encryptionProvider=org.bouncycastle.jce.provider`
  `.BouncyCastleProvider`

Boot using External Key

- **Db7 (AES)**
  - jdbc:derby:db7;
    encryptionAlgorithm=AES/CBC/NoPadding;
    encryptionKey=46894d733245130ed938230dd4019ec6
- **Must pass complete information used at create time**
- **Thus store algorithm and key in secure key store**

Boot & encryptionProvider

- **Encryption provider used to create database is not stored with database**
- **Is a runtime, environment setting**
- **A database created with a provider (e.g. BouncyCastle) should boot on a different or same JVM with another provider (e.g. SUNJCE)**
- **Assuming both support the same algorithm and key strength**
- **Thus if using specific provider must set on boot request**

Connecting to an Encrypted Database

- **Once database has been created or booted, any connection request can be made WITHOUT supplying the boot password or encryption key**
- **Connection requests are subject to authentication and authorization as usual**
- **Database remains booted after the initial connection is closed**
- **Not an issue for single user applications, e.g. on laptops**

Potential Improvement for Derby

- **Assuming external key store, define some Java interface the database engine calls back on to obtain the encryption key**
- **This would be called on every encrypt/decrypt operation, or maybe every N seconds**
- **If fail to get key, database is shutdown and purge in-memory data as much as possible**
- **E.g. An implementation would read the key from the smart card, if card is removed then shutdown**

Simpler Potential Improvement(s)

- **If no active connections to a database, shut it down**
- **If no activity in a database for N seconds, shut it down**

Changing Boot Password

- **When using database key store the boot password can be changed**
- **Only re-encrypts the key in service.properties, does not re-encrypt the complete database with a new key**
- **SQL Procedure**
  - `CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY( 'bootPassword', 'BU72x0tON65' , 'd8ErWe901nt');`
  - `VALUES SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY('bootPassword') will always return null`

Known Issue for Encrypted Databases

- **Derby.log is in clear text**
- **Some error messages include application supplied values, e.g. value too long for column.**
  - These will appear in derby.log in cleartext
  - Easiest fix is for error messages not to include values
- **Diagnostic properties can cause SQL text and parameter values to appear in error log**

Potential Starting Point for Crackers

- **On-disk files that map to system catalogs contain known data, thus may be good initial targets for crackers**
  - E.g. SYSTABLES contains text like SYSTABLES, SYSALIASES, SYSVIEWS etc.
- **May be others, e.g. anyone can create a Derby database to see what information is consistent**
- **Could only encrypt user tables, or tables selected by application**

Derby Authentication

User & Password Identification

- **Standard JDBC attributes in connection URL or Properties object**
  - user
  - password
- **User, Password parameters in DriverManager.getConnection() methods**
- **User, Password properties in DataSource**

Derby Authentication

- **Four types**
  - NONE - default
  - LDAP
  - BUILTIN
  - Application defined
- **Settable**
  - Per database
    - Set as database properties
  - For the system
    - Controls shutdown request

Default Authentication

- **NONE**
- **User name is not required to connect**
  - Defaults to APP if not supplied
  - Hence schema would be APP
- **Password not required**
- **User and password can be supplied**
  - User name will be default schema
  - Schema need not exist
  - Password ignored
- **Note – Network client (JCC) requires user and password**

LDAP

- **Set of properties**
  - `derby.connection.requireAuthentication=true`
  - `derby.authentication.provider=LDAP`
  - `derby.authentication.server=`*`ldap_server:389`*
  - *And optional set*
- **LDAP entries not cached by Derby**
- **Derby does not support LDAP groups**

BUILTIN

- **Set of properties**
  - `derby.connection.requireAuthentication=true`
  - `derby.authentication.provider=BUILTIN`
- **Users defined using properties**
  - `derby.user.`*`name`*`=`*`password`*
- **Database property**
  - `CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY( 'derby.user.dan', 'cindy')`
  - MD5 hash of password stored in database
  - As value of property derby.user.dan
- **System property**
  - Clear text password as system property

Application Defined

- **Set of properties**
  - `derby.connection.requireAuthentication=true`
  - `derby.authentication.provider=`*`java_class_name`*
- **Java class implements**
  `org.apache.derby.authentication.UserAuthenticator`
- **Single method**

```
public boolean authenticateUser(String userName,
  String userPassword, String databaseName,
  Properties info) throws SQLException
```

- **`info` contains contains connection attributes**
- **Returns**
  - true, user authenticated
  - false, failed authentication – 08004 SQL Exception
  - SQLException – failed authentication

Derby Authorization

Derby Authorization

- **GRANT/REVOKE not supported**
- **Simple per-database authorization scheme for a user name set by configuration properties**
  - User can connect and read & modify data
  - User can connect in read-only mode
  - User can not connect
- **May support GRANT/REVOKE in future**

Authorization Properties

- **Set default authorization for users**
  `derby.database.defaultConnectionMode=`
  `{ noAccess | readOnlyAccess | fullAccess }`
  - Default is fullAccess
- **Set full access authorization for specific users**
  `derby.database.fullAccessUsers=list_of_users`
- **Set read-only access authorization for specific users**
  `derby.database.readOnlyAccessUsers=list_of_users`
- **Authorization independent of authentication, user defined by user attribute on connection request**

Authorization Example

- **Limit read-write to sa**
- **Limit read-only to fred,ginger**
- **No other users allowed**
- **`derby.database.defaultConnectionMode=noAccess`**
  `derby.database.fullAccessUsers=sa`
  `derby.database.readOnlyAccessUsers=fred,ginger`

Database Properties

Derby Properties

- **Properties can either be in**
  1. System set (cmd line -D, System.getProperties)
  2. Application set (derby.properties file)
  3. Database (SYSCS_SET_DATABASE_PROPERTY)
- **Precedence order as above**
- **Thus, by default, a database's security settings can be overridden by properties in the JVM set or derby.properties**
- **Properties in 1,2 affect all databases and the Derby system when appropriate**

Database Only Property Setting

- **Set the database property derby.database.propertiesOnly=true**
- **Using**

```
CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY(
'derby.database.propertiesOnly', ‘true')
```

- **The database now will only use properties (set by `SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY`) to control its configuration & security settings**

Database Re-boot Required

- **After changing database authentication and/or authorization a database re-boot is required in order for them to take effect**
- **Re-boot by shutting down**
  - Conn.close();
  - DriverManager.getConnection(“jdbc:derby:db;shutdown=true”);
  - Reconnect with correct authentication

Configuration Warnings

- **It is possible to configure a database so no connections are possible**
- **There is no special system administrator password**
- **Examples**
  - Enable BUILTIN authentication with no users
  - Enable app defined authentication with an invalid class
  - Set default authorization to no access, with no users on the full access or read-only lists
- **Or only define read-users in a database, have access but cannot change anything**

Java 2 Security Manager

Java 2 Security Manager

- **Derby supports environments where the Java 2 Security Manager is enabled.**
- **Requires granting specific Java permissions to the Derby code**
  - E.g. read/write database files
- **Derby requires only the minimum permissions needed to perform its intended functionality as a database engine**

Derby Security Permissions (derby.jar)

- **Create class loaders – SQL queries are compiled to byte code and loaded by an internal class loader [Required]**
- **Read/Write permissions for data files [Required]**
- **Read derby.* System properties**
- **Read permission for derby.properties**
- **Read/Write permission for derby.log**
- **Install JCE provider**

Special Note for SQL Routines

- **When running with a Java 2 Security Manager Java functions and procedures must:**
  - Execute controlled actions using privileged blocks
  - Have permission for action granted to their code base (jar file)
    - Not possible at the moment if jar is stored in the database
- **The generated class that executes the SQL statement they are called from has no permissions and will be in the calling stack of the routine**
- **This means this procedure is not harmful when running with a security manager**
  - CREATE PROCEDURE SHUT_REMOTE_SYSTEM(e int) … EXTERNAL NAME 'java.lang.System.exit'
  - CALL SHUT_REMOTE_SYSTEM(-1)
  - Fails with a SecurityException

Wrap Up

Derby Security Summary

- **Complete disk encryption**
  - Application selected algorithm
- **Flexible authentication including LDAP**
- **Simple authorization**
- **Java 2 Security Manager key element**

Use Applicable Security

- **Applications can make use of these security features to provide an acceptable level of security for that application**
- **Examples**
  - Monitoring data could use DES encryption as data more than a day old no longer has value
  - A store kiosk could use no database security as data is public (catalog) and physical in-store security exists
  - A laptop home-banking application may require DESede with encryption key stored on a smart card

References

- **Derby & Security section in Derby Developers Guide**
- **http://java.sun.com/jce**
- **http://java.sun.com/security**
- **http://java.sun.com/jndi**