How to avoid storing credentials to connect to Oracle with JDBC?

Asked 16 years, 3 months ago Modified 7 years, 8 months ago

Viewed 15k times

28

Is it possible to setup a JDBC connection to Oracle without providing username/password information in a configuration file (or in any other standard readable location)?

Typically applications have a configuration file that contains setup parameters to connect to a database. Some DBAs have problems with the fact that usernames and passwords are in clear text in config files.

I don't think this is possible with Oracle and JDBC, but I need some confirmation...

A possible compromise is to encrypt the password in the config file and decrypt it before setting up the connection. Of course, the decryption key should not be in the same config file. This will only solve accidental opening of the config file by unauthorized users.

oracle-database **jdbc** **credentials**

Share

asked Sep 24, 2008 at 12:05

Improve this question

Follow

Hans Doggen
1,836 ● 2 ● 17 ● 13

See answer to this similar question:
stackoverflow.com/a/61429708/328275 – axiopisty Apr 25, 2020 at 17:32

11 Answers

Sorted by: Highest score (default)

6

You may want to try Kerberos which can use the OS user's credentials and adding the OS user to the database as identified externally. Make sure that you use Kerberos and not the old way of doing this, which had serious security issues.

For Kerberos support you would need the advanced security option and a recent JDBC driver, probably 11g version. Before trying to get it to work in Java, try it out in Sql*Plus using '/' as username and empty password. "select user from dual" should give you user@domain. You may also find that there is a fundamental difference between using thin or OCI driver when it comes to Kerberos configuration.

Share Improve this answer

Follow

answered Sep 27, 2008 at 10:42

stili
674 ● 6 ● 8

You definitely don't want to be able to connect to the database without credentials because that makes the

5

database vulnerable.

This is a general problem, how do I store credentials needed to access external systems? WebLogic has a credential mapper to solve this problem, in which credentials (encrypted) are stored in embedded LDAP. Many Oracle products use a credential store facility that stores credentials in Oracle wallet.

In the question, you provided the answer. Store the password encrypted and decrypt when you need it. Obviously you have to use symmetric encryption algorithm such as 3DES so you can decrypt it. Make sure the symmetric key is not something that can be guessed.

The trick is where you keep the symmetric key needed for en/de-cryption. You can put it in a file that is secured through the OS or you can keep it in the code, but then you need to keep the code secure. You can also generate the key if you use a technique that will produce the same key and the algorithm is reasonably secure.

If you can keep the code secure you can obviously keep the password in the code as well. However, you want the flexibility of being able to change the credentials without changing the code.

You can add more layers to this solution as well. You can encrypt the configuration file (with a different key) as well as the password inside it making the hacker discover 2 keys. There are other even more secure methods using PKI, but they get hard to set up.

Share Improve this answer

Follow

edited Sep 24, 2008 at 12:31

answered Sep 24, 2008 at 12:24

David G

6,327 ● 4 ● 34 ● 31

3

I'd suggest you look into proxy authentication. This is documented in the Oracle® Database Security Guide, as well as the Oracle® Database JDBC Developer's Guide and Reference. Essentially what this allows you to do is have a user in the database that ONLY has connect privileges. The users real database accounts are configured to be able connect as the proxy user. Your application connecting through JDBC then stores the proxy username and password, and when connecting provides these credentials, PLUS the username of the real database user in the connect string. Oracle connects as the proxy user, and then mimics the real database user, inheriting the database privileges of the real user.

Share Improve this answer

Follow

answered Sep 24, 2008 at 13:02

Mike McAllister

1,549 ● 2 ● 12 ● 15

Doesn't this approach make the security even worse? An attacker knowing the proxy username+password can access the database under any of those real accounts. That's why I offered using certificates (hopefully safely managed by the OS) instead of username+password for authenticating with the proxy. – Alexander Sep 24, 2008 at 13:35

You still need to secure the proxy username and password. But what it lets you do is control which users the middle tiers can connect as, and also the roles the middle tiers can assume for the user. – Mike McAllister Sep 24, 2008 at 13:42

Thanks for the information! I've never used this feature myself. Do you happen to know whether the JDBC layer can securely access OS-managed certificates. If this were possible, the method would be better than proxy username+password, because the cert would stick to the machine. – Alexander Sep 24, 2008 at 14:08

I'm not 100% sure, but this section of the documentation is your best bet at checking that out -> download.oracle.com/docs/cd/B19306_01/java.102/b14355/ … – Mike McAllister Sep 24, 2008 at 14:18

2

All `J2EE` containers (**JBOSS**, **Tomcat**, **BEA**) have connection pools. They will open a number of connections, keep them alive and will give them to you in `1/100`th the time it takes to create one from scratch.

Additionally, they also have cool features, in `JBOSS` for example, all the connection info is stored in an external file. If you change the connection info i.e., you switch from a *test* to a *production* DB, your application will dynamically be fed connections from the new pool

The good news is that you don't need to run a full `J2EE` container just to use connection pooling. The external resource allows the password to be stored in either plaintext, or pseudo-encrypted.

For a guide on using Tomcat's builtin connection pooling see the apache commons-dbcp:

- http://vigilbose.blogspot.com/2009/03/apache-commons-dbcp-and-tomcat-jdbc.html

Share Improve this answer

Follow

edited Aug 12, 2015 at 11:34

Mohammad Faisal

5,949 ● 15 ● 73 ● 124

answered May 4, 2009 at 21:50

Achille

466 ● 3 ● 8

1

To my knowledge jdbc connection usernames/passwords need to be stored as plain text. One way to limit the possible risks of this is to restrict the rights of the user so that only the given applications database can be used and only from a predefined host. IMO, this would limit the attacker very much: he could only use the un/pw from the same host where the original application resides and only to attack the original application's database.

Share Improve this answer

Follow

answered Sep 24, 2008 at 12:12

kosoant

11.6k ● 7 ● 33 ● 37

1 it's a violation of all standard security practices to store passwords as plain text. – David G Sep 24, 2008 at 23:54

1 Yes it's a problem, but what is then the solution in this case? – kosoant Dec 4, 2009 at 14:48

1

Have wondered this in the past.

The solution is certainly one that includes having proper network security at the server and network level to really reduce the number of people who can get access to the system, and having the database credentials only give access to a database account with the bare minimum of permissions required for the application to run.

Encryption of properties files might be enough of a deterrent in terms of "can't be bothered to find the key or passphrase" to get an attacker to go onto their next compromised server. I wouldn't rely on "my neighbour is less secure so steal from him please" security however!

Share Improve this answer

Follow

answered Sep 24, 2008 at 12:13

JeeBee

17.5k ● 5 ● 52 ● 60

1

There are two key approaches and both have a significant impact on the design of the system, such that it is not easy to move from one to the other without a significant rewrite. You need to understand what your companies security governance policy is before choosing.

1) Every user has credentials, that are carried through the application, for the service that is being used by the Application; in your case the Oracle database uses those user credentials to connect to the database. The downside is that every user needs a credentials for each secure service. This is a reasonable secure approach but also requires the signficant extra work to provide and maintain the user credentials. Your database administrators will need to actively manage user credentials, which may run counter to your company's security governance policies.

2) The Application database credentials are stored on a secure directory service, e.g. Secure LDAP. The Application accesses the directory service with the users' credentials. The directory service returns the approriate credentials for the service being accessed.

In both cases the database credentials should be limited to perform the appropriate operations only. The credentials should reflect the requirements of the business processes, for example; they allow select from defined views/tables, insert into others, but not create, update or drop tables. In the second approach use seperate credentials for each major business process, e.g. Order Processing, Accounting, HR, etc.

However remember that security is like layers of an onion, if somebody has stripped away the layers to access the application, such that they can access the DB contection pool config file. They can probably Trojan the

application to capture users' credentials. This is where a good security governance policy comes in.

Security Governance is a complex issue that needs senior management commitment, because if you need this level of security for your live platform, it costs. You need to separate responsibilities of development from deployment, operations & user authority management. You may also need to have security auditors, who have full access to view changes but no ability to change the configuration. It if far from simple and is highly paid specialism.

Share Improve this answer Follow

edited Sep 24, 2008 at 14:49

answered Sep 24, 2008 at 12:39

Martin Spamer

5,575 ● 30 ● 44

For context, this explanation assumes a discrete user community who will directly and personally interact with the Oracle database (such as business users in a corporation). That is in contrast to a common paradigm for web-based applications (which I think the author of the question was assuming) where an application or web server always connects to the database with the same credentials, and security concerns are dealt with via application code, not database-level access control. – Ian Varley Jun 10, 2010 at 0:53

1

Since I'm not entirely clear on your environment other than Java & JDBC talking to Oracle I'll speak towards that.

If you are talking about a Java EE app, you should be able to setup connection pools and data sources on the app server, then your application talks to the connection pool who handles connectivity at that level.

The connection pool and data source holds and secures the credentials.

Share Improve this answer

Follow

edited May 5, 2013 at 10:14

Arjan Tijms

38.2k ● 12 ● 111 ● 143

answered Sep 24, 2008 at 12:12

curtisk

20.2k ● 4 ● 60 ● 73

> Then how do you authenticate with the connection pool? Otherwise you have, essentially, no security. Might as well have no password on the oracle account either. – Ian Boyd Feb 24, 2010 at 18:48

1

In addition to the solutions that were already mentioned (Kerberos authentication, using proxy authentication) there are 2 other solutions that both work with the JDBC thin driver:

1. Store the password in an SSO wallet: a wallet can be used to store the user's password. If you use an SSO wallet then the wallet itself doesn't have a password. SSO wallets are commonly used in the context of SSL but they can also be used to just store a password.
2. Use SSL with user authentication: configure SSL with a user that's externally authenticated by the Distinguished Name (DN). This user doesn't have a password. As long as you connect with SSL using a certificate that has this DN you'll be able to create a session using this user.

Share Improve this answer

Follow

answered Apr 25, 2017 at 18:27

Jean de Lavarene

3,743 ● 1 ● 21 ● 30

1 option 2) is the best option, we are using this in my company to perform password-less authentication against Oracle databases (using certificates) to avoid using passwords and having to rotate them (you still need to renew the certificates on a regular basis though). This should be marked as the answer to the question. – olivierg Jun 3 at 19:27

Thanks @olivierg. Option 2) also provides very good performance. Note that Autonomous Database (in OCI) also offers token based authentication. – Jean de Lavarene Jun 10 at 15:20

You can store the credentials anywhere, including as hardwired strings in the program or as entries in the

0

Windows registry. It's up to you to retrieve them if you use something nonstandard, though; I'm not aware of any pre-rolled solutions that aren't plaintext.

Share Improve this answer

Follow

answered Sep 24, 2008 at 12:09

Hank Gay

71.8k ● 36 ● 161 ● 222

0

You could try Oracle's proxy authentication where the JDBC client authenticates using a certificate against a known middle-tier component/service (the proxy) which is trusted by the database server. I've never tried that though, so I don't know whether it's easy to do.

Share Improve this answer

Follow

answered Sep 24, 2008 at 12:18

Alexander

9,380 ● 2 ● 28 ● 23