DERBY-3532

djd

2008-03-13

Contents

Chapter 1

Root issue DERBY-3532

1.1 Summary

Invalid & possibly skipped authentication handling when shutting down the network server.

1.2 Description

In NetworkServerControlImpl.checkShutdownPrivileges() code fetches the internal authentication service to perform user authentication.

However if no such authentication service is found (null is returned) then authentication is bypassed, this has the potential of being a security hole.

The discussion in DERBY-2109 indicated that even with authentication NONE, there is still an internal authentication service, thus null is not a valid return when getting the internal authentication service. A secure fail safe system would be to not bypass authentication if null is returned.

I tried removing the check for null in the method and that lead to NullPointerExceptions. This means that something wrong is going on and very possibly no authentication checks are actually being made when shutting down the network server.

The null return might be due to checking the authentication after Derby has been shutdown.

1.3 Attachments

1. DERBY-3532.diff
2. ReproDerby3532.java
3. ReproDerby3532.java

1.4 Comments

1. **espinha:** Triaged for 10.5.2.Checked High Value Fix.
   Bumped urgency to Urgent.

   This seems like something we should fix for 10.5.2.

2. **kmarsden:** Changing to normal urgency. It is not a regression and nobody seems to be working on it so it won't make it into 10.5.2. I agree we should prioritize it for the next release.

3. **kristwaa:** Changed urgency to normal, as described in the previous comment.

4. **kmarsden:** Lily told me that she got a NPE in ServerPropertiesTest when she commented out the check Dan referenced. Lily, please try this program with your modified server to see if it pops the bug.

5. **lilywei:** I cannot reproduce AuthenticationService(auth) is null with the previous ReproDerby3532. I am able to reproduce AuthenticationService(auth) == null with the new repro file. When network service server is shutting down using embedded data source, network server does not know shutdown has been call. Hence, network server did not boot to fill the removed AuthenticationService.I am not sure what Derby can do at this case. It will be nice if derbynet.ServerPropertiesTest and management.InactiveManagementMBeanTest does not try to shutdown network server with embedded data source.

6. **kmarsden:** Lily said:
   I am not sure what Derby can do at this case. It will be nice if derbynet.ServerPropertiesTest and management.InactiveManagementMBeanTest does not try to shutdown network server with embedded data source.

   It is a little confusing with the different shutdowns going on.
   The tests are not shutting down network srever with the embedded driver but rather using it to shutdown the embedded engine which network server is using. So when we reproduce the bug the sequence is in a single JVM:
   1) start network server
   2) shutdown the embedded engine (not network server) with the embedded driver (either DriverManager.getConnection("jdbc:derby:;shutdown=true) or with EmbeddedDataSource as the latest repro.
   3) Shutdown network server. Since network server relies on the embedded engine to manage authentication and it is now gone the findService() call is null.
   I am sure also there are other network server cleanup issues in this scenario.

   In our first attempt at ReproDerby3532 in step 2 we shutdown the embedded engine with the client driver. In this case Network Server is aware that the engine is being shutdown because it came in through the client. Network Server catches the XJ015 error cleans up some of the network server threads and sessions and threads and restarts the embedded engine.
   This happens in DRDAConnThread.writeSQLCARD(). The cleanup and engine restart occurs in the poorly named NetworkServerControl.startNetworkServer() which cleans up network server if needed and starts the embedded engine.

   One option is to document that users should not shutdown the embedded engine with the embedded driver while network server is running and as you suggest, change here
   change the tests to shut down the engine with the client driver.
   I wonder instead if it would work and be worth it to put a lightweight check in network server for each client request to see if the embedded engine is running and if it is not, do the cleanup and start the embedded engine. Then we would always find the authentication service on shutdown and would be able to handle any other cleanup issues as well.

   Thoughts?

7. **lilywei:** I am hesitating to decide having lightweight check in network server for each client request to see if the embedded engine is running. If there is any good suggestion, I will be happy to try.
   I am trying to fix the test problem in junit regarding when network service server is shutting down it is using an embedded data source. First try, I change DriverManagerConnector.shutEngine and hope to shutengine with network server URL instead of embedded URL. However, if I use jdbcclient.getUrlBase(), I will get error like:
   Caused by: org.apache.derby.client.am.SqlException: The URL 'jdbc:derby://localhost:1527/' is not properly formed.
   Second try, If I take out the extra "/" from JDBCClient.DERBYNETCLIENT, I will get error:
   Caused by: java.sql.SQLException: Database '/localhost:1527/' not found.
   Is there any particular reason we put comment "Always shutsdown using the embedded URL thus this method will not work in a remote testing environment." in DriverManagerConnector.shutEngine()?" I think the writer is reading my mind now. I am open to any suggestion.

8. **kmarsden:** You got the exception org.apache.derby.client.am.SqlException: The URL 'jdbc:derby://localhost:1527/' is not properly formed
   It seems it is missing the shutdown. The url should be getURLBase() + ";shutdown=true'

   The comment is mysterious because1) We do not I think support remote testing with JUnit. 2) If we did, the embedded driver would not be available, just the client, so you wouldn't be able to shutdown using the embedded driver.

   I think it is ok to change

9. **lilywei:** Thank you, Kathey. I made the change with your suggestion.
   It is worth noting that Network Server will automatically restart Derby embedded if it detects in test environment. And, this is why management.InactiveManagementMBeanTest failed with the change. As Kathey pointed out, users can shutdown the embedded engine from client with ClientDataSource but not with DriverManager.getConnection(url) (please see DerbyRepro3532.java)
   If we allow ClientDataSource to shut down embedded engine, authentication service could be null. If DriverManager.getConnection(url) can not shutdown the engine, some tests will failed. i.e. management.InactiveManagementMBeanTest will still have embedded engine running since network server restart it. Maybe we should not allow ClientDataSource to shut down embedded engine and make it too heavy of check that can impact performance. Any suggestion is welcome.

10. **dagw:** Not sure if I understand this entirely, but as I read this thread, it is in some connection modes possible to shut down the engine underneath the network server, in other modes it is not possible?If so, it seems wrong that it should be possible to take down the engine when the server is running. It seems to me, the correct sequence is to first take down the server, then the engine.

11. **kmarsden:** Dag said:
    >it is in some connection modes possible to shut down theengine underneath the network server, in other modes it is not possible?

Yes currently from the client you can shutdown the engine remotely with ClientDataSource but not with ClientDriver/DriverManager. Thetwo should be the same. The question is whether to to disable this capability for ClientDataSource or enable it for ClientDriver. Disabling functionality of course has the risk of regressing someone that is using it.

If running in the same JVM you can also shutdown the engine with EmbeddedDriver or EmbeddedDataSource. I think for these two we just should document that it is ill advised. I don't think we can prevent it.

12. **lilywei:** I create DERBY-4345 for the community to decide whether to disable the capability for ClientDataSource to shutdown embedded engine or enable ClientDriver for such capability.
    At this point, maybe we could document and suggest users should not shutdown the embedded engine with the embedded driver while network server is running. Otherwise, Network Server failed to catch XJ015 error cleans up network threads and sessions and restarts the embedded engine. Therefore, they might not have a authentication handle.

13. **dagw:** I think we have now established that one can shut down the engine from the client using both ClientDataSource and the DeriverManager (see DERBY-4345). The server detects this. The question is what should we do, if anything, about the fact that using the embedded driver (EmbeddedDataSource or driver manager), we can shut down the engine underneath the server.

    Kathey> I wonder instead if it would work and be worth it to put a
    Kathey> lightweight check in network server for each client request to
    Kathey> see if the embedded engine is running and if it is not, do the
    Kathey> cleanup and start the embedded engine. Then we would always
    Kathey> find the authentication service on shutdown and would be able
    Kathey> to handle any other cleanup issues as well.

    I think it would be good if the network server were able to detect the shutdown and cleanup/reboot the engine, so we get the same behavior using embedded and client shutdown. Would it be costly to have such a check?

14. **kmarsden:** Unchecking HVF. Looks like this is a sticky issue with possible compatibility concerns

Chapter 2

Connected issue DERBY-2109

2.1 Summary

System privileges

2.2 Description

Add mechanisms for controlling system-level privileges in Derby. See the related email discussion at http://article.gmane.org/gmane.comp.apache.db.derby.devel/33151.

The 10.2 GRANT/REVOKE work was a big step forward in making Derby more secure in a client/server configuration. I'd like to plug more client/server security holes in 10.3. In particular, I'd like to focus on authorization issues which the ANSI spec doesn't address.

Here are the important issues which came out of the email discussion.

Missing privileges that are above the level of a single database:

- Create Database
- Shutdown all databases
- Shutdown System

Missing privileges specific to a particular database:

- Shutdown that Database
- Encrypt that database
- Upgrade database
- Create (in that Database) Java Plugins (currently Functions/Procedures, but someday Aggregates and VTIs)

Note that 10.2 gave us GRANT/REVOKE control over the following database-specific issues, via granting execute privilege to system procedures:

Jar Handling
Backup Routines
Admin Routines
Import/Export
Property Handling
Check Table

In addition, since 10.0, the privilege of connecting to a database has been controlled by two properties (derby.database.fullAccessUsers and derby.database.defaultConnectionMode) as described in the security section of the Developer's Guide (see http://db.apache.org/derby/docs/10.2/devguide/cdevcsecure865818.htm

2.3 Attachments

1. DERBY-2109-02.diff
2. DERBY-2109-02.stat
3. derby-2109-03-javadoc-see-tags.diff
4. DERBY-2109-04.diff
5. DERBY-2109-04.stat
6. DERBY-2109-05and06.diff
7. DERBY-2109-05and06.stat
8. DERBY-2109-07.diff
9. DERBY-2109-07.stat
10. DERBY-2109-08_addendum.diff
11. DERBY-2109-08_addendum.stat
12. DERBY-2109-08.diff
13. DERBY-2109-08.stat
14. DERBY-2109-09.diff
15. DERBY-2109-09.stat
16. DERBY-2109-10.diff
17. DERBY-2109-10.stat
18. DERBY-2109-11.diff
19. DERBY-2109-11.stat
20. DERBY-2109-12.diff
21. DERBY-2109-12.stat
22. SystemPrivilegesBehaviour.html
23. systemPrivs.html
24. systemPrivs.html
25. systemPrivs.html
26. systemPrivs.html

2.4 Comments

1. **rhillegas:** Added "Boot all database" as a system privilege based on Dag's feedback on the email thread. "Encrypt database" is already included in the list, based on feedback from Francois.

2. **rhillegas:** Added "Upgrade database" also per Dag's comments.

3. **djd:** Boot all databases is not a security issue for a client/server configuration. There is no mechanism for a client to enable or disable it.

4. **rhillegas:** Removed "Boot all databases" from system-wide list based on Dan's comment.

5. **rhillegas:** Attaching first rev of a functional spec for this feature. This spec incorporates feedback from the discussion on derby-dev:

   o Proposes using Java security mechanism to enforce system-wide privileges

   o Proposes restricting database-wide privileges to the Database Owner

   Would appreciate your continued feedback. Thanks.

6. **djd:** I don't think the "plugin" privilege is a system privilege since (as described in systemPrivs.html) it is restricting the ability to perform a database operation. I also think that the "plugin" privilege as described of being the privilege of being able to call CREATE FUNCTION or PROCEDURE is not the correct approach. The actual issue is being able to create a Java routine that maps to a class name not stored in a jar file.

   http://mail-archives.apache.org/mod_mbox/db-derby-dev/200611.mbox/%3c456C88DC.8080009@apache.org%3e

7. **davidvc:** Hi, Rick, nicely done! Here are my comments:

   - I like the use of Java security for system privileges
   - I think the system-wide permissions defined in the DerbyPermissions class can be a little more descriptive, so that they are clear without having to refer to documentation:
   o "create-plugin" rather than just "plugin"
   o "shutdown-engine" rather than just "shutdown" (could be confused with permission to shut down a db)
   o "create-database" rather than just "create" (create what?)

   "We don't see why anyone other than the database owner would need to shutdown, upgrade or encrypt that database." Hm, why not? Why wouldn't I want to grant that privilege to others besides myself? I always get nervous when someone says "I can't see why anyone would ever want to do this." I would rather we explicitly say we're not doing this for *now*, but not make hardcoded assumptions that can leak their way into our code...

   Similarly, "It's hard to imagine why you would want to grant more than one person the power to shut down the engine." Hm, there's that phrase again :). I would prefer to think of "system administrator" as a *role* rather than a *person*. And it seems reasonable to me that you may want to grant more than one person system administrator rights. At a minimum our architecture shouldn't assume only one system administrator per database engine.

   "there is no way to change ownership of a database" - is this a hardcoded fact of the Derby architecture, or is it something that's just not supported right now but could be enabled in the future? Also, please clarify you mean an operating system account when you say "it may be prudent to create a special account for this.'

What does '???' mean for "Documentation." It would be good for you to describe what documentation changes/additions will be needed for this feature.

Thanks,

David

8. **djd:** Rick thanks for writing this up and looking & proposing how to use the Java SecurityManager.

   I would not have the fake concept of a " System Administrator ". With descriptions like this it's better to be precise in the functional specification, maybe the user documentation could explain it in more detail using the concept of a System Administrator.
   e.g. in the functional spec replace sentances like this:

   If the engine runs under a SecurityManager, then only a System Administrator can halt the engine.

   with the technially correct version of:

   If the engine runs under a SecurityManager, then the shutdownEngine DerbyPermission is required.

   Since I assume since these are Java permissions that they can be granted to code as well as Principles or are they restricted in some way? E.g. can I grant shutdown engine to all code?

   I would not use DerbyPermission for the class name, the class name should describe permissions it is covering, maybe
   org.apache.derby.security.SystemPermission ?

   The code should not assume the Princple implementation is DerbyPrinciple, seems like it would be useful to be able
   to use other implementations of Principle. E.g. the existing UserAuthenticator class could be expanded to have a new method
   public Principle getPrinciple(String userName)
   allowing use of existing implementations such as X500Principal or the Principle implementation returned from an LDAP setup.

   Similar naming concerns for DerbyPrinciple, the class is already in the derby namespace so it should be obvious that it's a Derby user, I think it's really something like:
   org.apache.derby.authentication.DatabasePrinciple ??

9. **djd:** Could you add the reasons why Derby will require the "doAsPrivileged" permission and not just the ""doAs" permission?

   J2ME/CDC/Foundation 1.0 doesn't support the javax.security.auth.AuthPermission (or any javax.security class),
   any thoughts on what the functionality will be in J2ME?

10. **rhillegas:** Thanks Dan and David for your quick feedback. I will incorporate your suggestions into the next rev of the spec:

    1) Improve the package, class, and privilege names as suggested.

    2) Avoid talking about what I can't imagine.

    3) Say a bit more about database ownership: I don't see any architectural problem with changing the ownership; if someone wants to build this add-on feature, they can.

    4) Replace the '???' with a sentence or two stating that I haven't identified all the bits of documentation which need to be changed but that this is something we need to do before closing this issue.

    5) Avoid creating the impression that there is an architecturally significant System Administrator role.

    6) I believe it is correct that you can grant the system permissons to code as well as users. I will clarify this.

    7) I like the flexibility of adding a getPrinciple() method to UserAuthenticator. One question: do our stability conventions let us change an old customer-implemented interface in this way?

    Dan, could you say something more about how you think we should sand-down the plugin privilege:

    8a) I agree that one of the problems is the ability to invoke code outside the jar files supplied with the application. But I think there are other issues. For instance, there may be publicly accessible methods in the application jar files which should not be called without setting up some context.

    8b) Are you saying that you think this privilege should be scoped per database rather than per system? If so, do you think we should invent GRANT/REVOKE syntax for this? Or add a database-scoped Java permission which can be granted in the policy file?

    Thanks,
    -Rick

11. **rhillegas:** Dan, here are some responses to your last set of questions yesterday:

    1) Why doAsPrivileged() rather than doAs(): The first method is the one which is recommended for server environments (search for doAsPrivileged in the following webpage: http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html).

    2) J2ME/CDC/Foundation 1.0: The system-wide privileges seem to be important at the server site for client-server configurations. I don't see J2ME/CDC as a server platform. Would it be acceptable to say that system-wide privileges are only meaningful on the larger, client-server platforms?

12. **djd:** doAsPrivileged() vs. doAs()
    Thanks for the link to the JAAS reference, I think a better technical justification is needed than "it's recommended", especially since I don't see any such recommendation, I only see a comment saying one example where it might be useful is for servers.

    Maybe you could walk through an example when the call stack looks like:

framework code -calling» application code -calling» Derby code

and the application code is requesting a create database and the policy file is setup to allow rick to perform create database and the connection request is made as rick.
Why in this case is doAsPrivileged() required or better than doAs?

13. **oysteing:** Thanks for taking on this work Rick. I think it is very important improve such security issues in order to have a good story with respect to deployment of a Derby network server. Here are my comments/questions to the document:

    * I think my biggest concern with the proposed scheme for system-wide privileges is that it seems to fall into the same category as Derby's static properties. That is, you will have to restart the server in order for changes to take effect. While this is OK as a starting point, how difficult will it be to extend this to allow privileges to be granted and revoked without having to restart the server? Does the Java security framework have any support for run-time changes?

    * While it may be true that the Java Security framework might be familiar to users who currently use the Security Manager, I am not sure that is the case for the average users that would like to use system privileges. I think a lot more people will be more concerned about applications shutting down the server by accident, than that applications may write stuff where they should not. Your proposal is not particulary familiar to DBA's that have experience from other RDBMS like MySQL, Postgres, Oracle, or DB2. However, you can argue that the familiarity issue is much larger with the way Derby does user management so that this does not add much to the existing problem. :-)

    * I do not understand why plugins are a system-wide features. Are we not talking about objects that are local to a specific database?

    * I miss a rationale for how you ended up with three database-specific features. What are particular to them compared to other features like those listed in Appendix A?
    * As others, I react to statements like "We don't see why anyone other than the database owner would need to ...". I can see many reasons to the contrary. I guess want you mean is "We think it is an acceptable restriction that only the database owner is allowed to ...". I think the whole problem here is that database owner is really a role and not a particular person, and that we need to be able to define roles in order to make this really usable.

    * You say that the above can be solved by creating special accounts for the system/database administrator roles. At the same time we advertising the usefulness of pluggable user authentication. I do not feel these stories fit well together. You may have limited freedom to create specific users if you depend on

external user authentication.

* While all database-specific privileges is lumped into one
user/role, the system administrator privileges can be specified
one by one. Is there any particular reason for the finer
granularity for system privileges?

14. **dagw:** ÃŸystein:
> I miss a rationale for how you ended up with three
> database-specific features. What are particular to them compared
> to other features like those listed in Appendix A?

I think the reason is that these can all be controlled via execute
privileges on existing system procedures, whereas, the three new ones
are done via getConnection and are not currently protectable. It would
be nice to call this out earlier in the write-up - it is mentioned in the
beginning of appendix A.

15. **djd:** I added a wiki page about Java routine security, writing it all out helped my thoughts on the issues.

http://wiki.apache.org/db-derby/JavaRoutineSecurity

Taking Rick's idea of treating the CLASSPATH token as a pseudo jar, SYS.CLASSPATH, a little further I think the security issues can be aligned with the standard grant/revoke USAGE on a jar file, thus not requiring any special apis. See the wiki page for details.

I also think that Java routine security should be a seprate Jira issue to this one which is for System priviliges.

16. **rhillegas:** Thanks for your comments, ÃŸystein. A couple responses follow:

1) Dynamically changing the policy file. The policy file can be dynamically reloaded after you edit it and without restarting the VM. Some code has to call java.security.Policy.getPolicy().refresh(). This requires granting that code the following permission:

java.security.SecurityPermission "getPolicy"

Thanks for broaching this topic. We could always reload the policy file just before checking for a system privilege and then advise the user to grant derby.jar the above permission.

2) Unfamiliar api. Oracle, DB2, Postgres, and MySQL all handle system privileges in different ways. Picking one of these models would still result in an api that's unfamiliar to many people. That said, these databases do tend to use GRANT/REVOKE for system privileges, albeit each in its own peculiar fashion. I agree that GRANT/REVOKE is an easier model to learn than Java Security. I think however, that the complexity of Java Security is borne by the derby-dev developer, not by the customer. Creating a policy file is very easy and our user documentation gives simple examples which the naive user can just crib. With adequate user documentation, I think this approach would be straightforward for the customer.

3) Plugin scope. I think that you and Dan agree that "plugin" is a database-specific, not

a system-wide privilege. My first reaction (still recorded in the description block for this JIRA) also listed "plugin" as a database-wide privilege. I can argue the issue both ways. On the one hand, the "plugin" power potentially gives the user the ability to expose/exploit code which has system-wide effects. On the other hand, the affected objects (jars, functions, procedures) are all scoped to the database level. I am happy to treat "plugin" as a database-specific privilege.

4) Appendix A issue. I believe that Dag answered this one. Thanks, Dag!

5) My imagination deficit. As I said to David, I will reword these sentences.

6) User management and role migration. I suspect that the introduction of SQL roles will help address your concerns about database-specific privileges: we should be able to introduce a DB_OWNER role. In addition, as the spec notes, a follow-on usability feature could introduce the power to change the owner of a database. I think we're in better shape for system-wide privileges managed by Java security. Here the system administrator can edit the policy file as necessary.

7) Granularity of system-wide privileges. I see no problem in starting out lumping all the database-specific privileges together and letting the database owner hoard them. They are all privileges which a database owner would want. The two system-wide privileges belong to different roles: shutdownEngine belongs to the system administrator and createDatabase belongs to the various database owners. That is the motivation for the granularity of system-wide privileges.

17. **djd:** "The two system-wide privileges belong to different roles: [snip] ... and createDatabase belongs to the various database owners."

    That comment would seem to imply that create database permission needs to have a path associated with it.

    permission org.apache.derby.security.SystemPermission "${derby.system.home}${/}fred" "createDatabase" ;

    providing control over where each user can create databases.

18. **djd:** " We could always reload the policy file just before checking for a system privilege and then advise the user to grant derby.jar the above permission. "

    Not sure that's a good idea, seems like the refresh policy should be an explict operation, not called on every security operation just because it might have changed. Such an approach could allow some form of denial of service attack because a failed attempt to do something still would call refresh, thus repeated calls to the disallowed operation would still have a potential effect on the VM. Possibly stalling it as the policy file is reparsed.

19. **oysteing:** Rick Hillegas (JIRA) wrote:
    > [http://issues.apache.org/jira/browse/DERBY-2109?page=comments#action_12458313

]> > Rick Hillegas commented on DERBY-2109:
> ----------
» Thanks for your comments, ÃŸystein. A couple responses follow:
» 1) Dynamically changing the policy file. The policy file can be
> dynamically reloaded after you edit it and without restarting the
> VM. Some code has to call
> java.security.Policy.getPolicy().refresh(). This requires
> granting that code the following permission:
» java.security.SecurityPermission "getPolicy"
» Thanks for broaching this topic. We could always reload the policy
> file just before checking for a system privilege and then advise the
> user to grant derby.jar the above permission.
>
Good, and I agree with Dan that a command for explicit reloading is
probably best.

> 2) Unfamiliar api. Oracle, DB2, Postgres, and MySQL all handle
> system privileges in different ways. Picking one of these models
> would still result in an api that's unfamiliar to many
> people. That said, these databases do tend to use GRANT/REVOKE
> for system privileges, albeit each in its own peculiar fashion. I
> agree that GRANT/REVOKE is an easier model to learn than Java
> Security. I think however, that the complexity of Java Security
> is borne by the derby-dev developer, not by the
> customer. Creating a policy file is very easy and our user
> documentation gives simple examples which the naive user can just
> crib. With adequate user documentation, I think this approach
> would be straightforward for the customer.
>
I guess most people should be able to manage the editing of files etc.
I foresee that some users will be a bit confused about the location of
such files, but that is a more general issue.

> 3) Plugin scope. I think that you and Dan agree that "plugin" is a
> database-specific, not a system-wide privilege. My first reaction
> (still recorded in the description block for this JIRA) also
> listed "plugin" as a database-wide privilege. I can argue the
> issue both ways. On the one hand, the "plugin" power potentially
> gives the user the ability to expose/exploit code which has
> system-wide effects. On the other hand, the affected objects
> (jars, functions, procedures) are all scoped to the database
> level. I am happy to treat "plugin" as a database-specific
> privilege.

I see your point, but I do not think many organizations will find it
practical to deny database owners to be able to create plugins.
Hence, you might as well make it a database-specific feature.

» 4) Appendix A issue. I believe that Dag answered this one. Thanks, Dag!

Yes, and his answer made me realize that restore is missing from this
spec.

» 5) My imagination deficit. As I said to David, I will reword these sentences.
>
Great.

> 6) User management and role migration. I suspect that the
> introduction of SQL roles will help address your concerns about
> database-specific privileges: we should be able to introduce a
> DB_OWNER role. In addition, as the spec notes, a follow-on
> usability feature could introduce the power to change the owner
> of a database. I think we're in better shape for system-wide
> privileges managed by Java security. Here the system
> administrator can edit the policy file as necessary.

I agree. We need provide roles in order make this really usable.

> 7) Granularity of system-wide privileges. I see no problem in
> starting out lumping all the database-specific privileges
> together and letting the database owner hoard them. They are all
> privileges which a database owner would want. The two system-wide
> privileges belong to different roles: shutdownEngine belongs to
> the system administrator and createDatabase belongs to the
> various database owners. That is the motivation for the
> granularity of system-wide privileges.

In other words, for createDatabase you provide the list of potential database owners. But then in order to allow one person to create one database in one specific place, you allow him to create as many databases he wants wherever he likes. An alternative could be to only allow System Administrators to create databases, and provide a way to specify database owner when you create the database.

20. **djd:** » 3) Plugin scope. I think that you and Dan agree that "plugin" is a
» database-specific, not a system-wide privilege. My first reaction
» (still recorded in the description block for this JIRA) also
» listed "plugin" as a database-wide privilege. I can argue the
» issue both ways. On the one hand, the "plugin" power potentially
» gives the user the ability to expose/exploit code which has
» system-wide effects. On the other hand, the affected objects
» (jars, functions, procedures) are all scoped to the database
» level. I am happy to treat "plugin" as a database-specific
» privilege.
» I see your point, but I do not think many organizations will find it
> practical to deny database owners to be able to create plugins.
> Hence, you might as well make it a database-specific feature.

    I would actually say you aleady have the system level ability, because the actions of any java code is controlled by the installed security manager and policy which is set at a system level. Thus the ability to create Java routines should be controlled at the datbaase level and the ultimate power of those routines is controlled at the system level.

21. **rhillegas:** I do not understand the need to qualify createDatabase privilege with a list of databases which the user is allowed to create. This is not how other vendors manage this privilege. DB2, Oracle, and Postgres all have a generic createDatabase privilege. If you have this privilege, then you can create as many databases as you want. I think that the

DB2 documentation is most clear about the danger associated with this privilege. This is discussed in the DB2 SQL Reference manual, volume 1, in the section on "Authorization and privileges". For DB2, the ability to create databases is a superpower granted to the SYSCTL role and the point is to control who can consume system resources, including the power to consume disk space. In general, whether you can consume disk space is a consequence of the ability to create a database and is not refined by how many databases you can create.

I don't understand what additional security hole is plugged by qualifying the createDatabase privilege with a list of database names. I recommend that we follow the lead of these other vendors and implement this as a general privilege whose purpose is to control the power to consume disk space.

22. **djd:** The permission would not be the names of databases that one can create, but instead locations where databases can be created.
This has been requested in the past. If one wanted to match the other databases then the path could be very liberal.
Thus we can match existing databases and optionally provide a better security model because Java allows it.

23. **rhillegas:** I would like to continue the discussion of using doAsPrivileged() vs. doAs(). I ran the following experiments. For more detail on the code, please refer to the example code in Appendix B of the attached functional spec.

    EXPERIMENT 1:

    a) I separated the ShutdownAction into its own class. This is the PrivilegedExceptionAction which actually calls checkPermission(). I put this class in jarfile jar_A along with the custom Principal and Permssion classes.

    b) I put the entry point class in its own jar file jar_B. This class called the ShutdownAction using both doAs() and doAsPrivileged()

    c) I created a policy file which did the following:

    i) gave jarfile_B the privileges doAs and doAsPrivileged

    ii) gave Shutdown privilege to one distinguished Principal

    I observed the following:

    I) Calling doAs() failed for all Principals. That is, checkPermission() always reported that the Shutdown privilege was not granted. This is because code from jarfile_B was at the top of the stack but jarfile_B did not have Shutdown privilege.

    II) Calling doAsPrivileged() functioned correctly: It verified that the distinguished Principal had the Shutdown privilege and that no-one else did. This is because the call to doAsPrivileged() passed in a null AccessControlContext. This essentially told the security subsystem to not bother checking permissions for stack frames above the caller.

    EXPERIMENT 2

    This was the same as the previous experiment except that the policy file gave Shutdown privilege to jarfile_B. I observed the following:

I) doAs() now functioned correctly. The checkPermission() call verified that only the distinguished Principal had Shutdown privilege.

II) doAsPrivileged() continued to function correctly as in the previous experiment.

This suggested that if we wanted to use doAs(), then we would need to split derby.jar into 2 ProtectionDomains. The outer ProtectionDomain would have to be granted Shutdown privilege. Code in the inner ProtectionDomain would be run as a PrivilegedExceptionAction under the identity of the authenticated authorizationId. This inner domain would not be granted a blanket Shutdown privilege--this is what would allow checkPermission to distinguish the privileges of Principals who were trusted with Shutdown power.

EXPERIMENT 3

Here I tried to model an application embedding Derby. This was identical to the previous experiment with the following additions:

d) A thin entry point (main class) was created and put in its own jarfile_C. All that this class did was call the entry point in jarfile_B.

e) The policy file was enhanced to grant doAs and doAsPrivileged to jarfile_C.

This experiment behaved like the first experiment. That is:

I) doAs() failed always because the outermost ProtectionDomain, jarfile_C, did not have Shutdown privilege.

II) doAsPrivileged() correctly detected that only the distinguished Principal had Shutdown privilege. This is because doAsPrivileged() threw away all of the outer ProtectionDomains, including the untrusted jarfile_C domain.

CONCLUSIONS

1) Using Java Security to police engineShutdown and createDatabase will mean that additional privileges (doAs or doAsPrivileged) must be granted not just to derby.jar but to all ProtectionDomains above Derby on the call stack.

2) Using doAs means that we will have to factor derby.jar into two ProtectionDomains.

24. **dagw:** > CONCLUSIONS
» 1) Using Java Security to police engineShutdown and createDatabase
> will mean that additional privileges (doAs or doAsPrivileged) must
> be granted not just to derby.jar but to all ProtectionDomains above
> Derby on the call stack.

Can't this be avoided by running authentication and doAs/doAsPrivileged inside a plain doPrivileged's PrivilegedAction? Then we just need to allow derby.jar the "doAs/doAsPrivileged" permission? Or is there something special about these permissions?

25. **rhillegas:** Thanks, Dag. You are right. If the Subject.doAsPrivileged() call itself is run under an AccessController.doPrivileged() call, then you don't need to grant "doAsPrivileged" permission to the customer code. This makes the policy file simpler. All you need is:

    1) to grant "doAsPrivleged" permission to derby.jar

    2) to grant specific privleges (like Shutdown) to specific Principals.

26. **rhillegas:** I have attached a second rev of the functional spec for system privileges. This rev attempts to address the feedback on the first rev. Major changes between the revs are summarized in the introductory table of versions. I would appreciate the community's continued feedback.

27. **djd:** Thanks for incorporating the various comments into the new spec, it's looking better, a few comments.

    Java convention would have the target names for the permissions being "shutdownEngine" and "createDatabase"

    "org.apache.derby.security.FilePermission - This class extends the JRE's FilePermission."
    java.io.FilePermission is final.
    Seems strange to have these system permissions split across two classes, also databases are not restricted to files, so why "FilePermission"?

    Is this spec meant to cover & define the work for DERBY-2196? In some cases it seems to be (e.g. section on Network Server) but doesn't completely describe the changes required by DERBY-2196.

28. **rhillegas:** Thanks for the quick feedback, Dan. A couple responses follow:

    1) The camel-case permission names you suggest are fine by me.

    2) You're right, FilePermission is final so it can't be extended directly. The intention was to re-use its implies() logic, but that can be done via other means.

    3) I split the Derby permission into 2 classes due to an inordinate fondness for modelling this on java.security.BasicPermission. I can abandon this attachment to BasicPermission in favor of something like this:

    permission org.apache.derby.security.SystemPermission "shutdownEngine";
    permission org.apache.derby.security.SystemPermission "${derby.system.home}${/}accountingDBA" "createDatabase";

    4) This spec does not cover and define the work for DERBY-2196. A separate spec has to be written for that JIRA. For the moment, let's imagine a spec for DERBY-2196 which follows the outlines of the wiki page linked from that issue. This spec (for DERBY-2109)

then adds a delta on top of DERBY-2196: the addition of the Derby SystemPermissions to the Basic policy.

29. **djd:** Rick Hillegas (JIRA) wrote:

    > 3) I split the Derby permission into 2 classes due to an inordinate fondness for modelling this on java.security.BasicPermission. I can abandon this attachment to BasicPermission in favor of something like this:
    » permission org.apache.derby.security.SystemPermission "shutdownEngine";
    > permission org.apache.derby.security.SystemPermission "${derby.system.home}${/}accountingDBA" "createDatabase";

    Once you mention BasicPermission the split makes sense (ie. shutdownEngine does not have an action), so adding more text around this in the functional spec would be good. More along technical lines than "a fondness for" though :-)

    Seems like the permission class for "createDatabase" should be DatabasePermission, I could see this in the future being expanded to have additional actions of "shutdown", "drop", "encrypt" etc. Then of course the "createDatabase" action could be "create".

    > 4) This spec does not cover and define the work for DERBY-2196. A separate spec has to be written for that JIRA. For the moment, let's imagine a spec for DERBY-2196 which follows the outlines of the wiki page linked from that issue. This spec (for DERBY-2109) then adds a delta on top of DERBY-2196: the addition of the Derby SystemPermissions to the Basic policy.

    Ok - I think I get it. You writing this assuming that DERBY-2196 is done and doesn't control system shutdown or database creation, and then this spec describes how that basic policy would be changed by these permissions. Right? Just hadn't thought of that order. :-)

    One issue I see in the network server section is that it assumes a mapping between the Java (OS) user name and the database user name. This would be a first for Derby, no where else is such an assumption made.
    I.e. ${user.name} is never used elsewhere in the code.
    Not sure of the implications of this, or if there is a better user name to grant shutdown system to for the basic policy.

30. **djd:** Daniel John Debrunner (JIRA) wrote:

    > Seems like the permission class for "createDatabase" should be DatabasePermission, I could see this in the future being expanded to have additional actions of "shutdown", "drop", "encrypt" etc. Then of course the "createDatabase" action could be "create".

    Just to add to this, it would be wise to define the format for the database location carefully to ensure the model works for future enhancments, such as the ability to limit shutdown of a database in a jar file, or creation of some non-file based database.

    In the spec it says:
    "The directoryLocation argument has the format of a directory reference in a declaration of a FilePermission permission. "

    Databases can be more than files though, should the target for this permission be in the format of a database name from a JDBC URL or DataSource databaseName property?

31. **rhillegas:** Daniel John Debrunner (JIRA) wrote:
>Ok - I think I get it. You writing this assuming that DERBY-2196 is done and doesn't control system shutdown or database creation, and then this spec describes how that basic policy would be changed by t>hese permissions. Right? Just hadn't thought of that order. :-)

Right, that's what was in my brain area.

>One issue I see in the network server section is that it assumes a mapping between the Java (OS) user name and the database user name. This would be a first for Derby, no where else is such an >assumption made.
I>.e. ${user.name} is never used elsewhere in the code.
>Not sure of the implications of this, or if there is a better user name to grant shutdown system to for the basic policy.

Freewheeling some other alternatives which have various drawbacks:

1) The Basic policy could grant shutdownEngine to everyone. Seems insecure.

2) We could add an optional argument to the server startup command, letting the customer specify the userName of a Principal who should have shutdownEngine power. This would not be useful if the customer had their own Principal implementation.

3) We could add another method to UserAuthenticator. That method would return an array of Principals who should have shutdownEngine power. If using the builtin UserAuthenticator, this would default to just the DatabasePrincipal named ${user.name}.

Your issue also made me realize that the spec needs to say something about what the Basic policy should be for createDatabase privilege. Here are some suggestions:

A) This could be the same user(s) who have shutdownEngine privilege.

B) This could be granted to everyone.

32. **rhillegas:** Daniel John Debrunner (JIRA) wrote:

>Just to add to this, it would be wise to define the format for the database location carefully to ensure the model works for future enhancments, such as the ability to limit shutdown of a database in a jar file, >or creation of some non-file based database.
>
I>n the spec it says:
">The directoryLocation argument has the format of a directory reference in a declaration of a FilePermission permission. "
>
>Databases can be more than files though, should the target for this permission be in the format of a database name from a JDBC URL or DataSource databaseName property?

Maybe we could require that the database location be unambiguously qualified by one of the subsubprotocols which we recognize on URLs:

classpath:
directory:

jar:

So, for instance, in this first rev:

permission org.apache.derby.security.DatabasePermission "directory:${derby.system.home}${/}accountingDBA" "create";

and in some later rev when we support other kinds of DatabasePermissions:

permission org.apache.derby.security.DatabasePermission "jar:/maps/USmaps" "shutdown";

and if we added a new subsubprotocol for in-memory databases:

permission org.apache.derby.security.DatabasePermission "memory:/users/fred/-" "create";

33. **djd:** Rick wrote:

> Maybe we could require that the database location be unambiguously qualified by one of the subsubprotocols which we recognize on URLs:
» classpath:
> directory:
> jar:
» So, for instance, in this first rev:
» permission org.apache.derby.security.DatabasePermission "directory:${derby.system.home}${/}accountingDBA" "create";

That's possible though it would be good to see if the disambiguating rules that are used for database name could also be used for database location, though as a first step requiring "directory:" would be ok and forward compatible.

If the format of database location is to follow database name (with extensions) then it's important to note that the separator for database names is always forward slash since they are part of a URL syntax (JDBC URL). Though your example has a variable with a file system name in it, (and others such as user.dir and user.home would be useful) which can contain a different separator, it does seem this is a useful feature. So the spec should account for this.

We should also support relative names, as in database name, thus your example could be re-written as:

permission org.apache.derby.security.DatabasePermission "directory:accountingDBA/*" "create";

The extensions to the database format are the use of '/-', the functional spec describes it but the description is inconsistent compared to existing use in the policy file. I believe that the correct usage should be:

directory:name - defines that specific single database identified by name
directory:name/* - defines permission on any database in the folder described by name
directory:name/- - defines permission on any database in the folder or sub-folder of the folder described by name

I think this is more flexible, e.g. the future ability to grant shutdown database to a single database, and matches FilePermission which you are intending to use to imlement it.

So a quick summary would be I believe consistency with existing practices is very important, I think it will reduce the number of errors in setting up policy files by users, which are tricky things.

34. **djd:** The functional spec says:

We add a new method to Derby's org.apache.derby.authentication.UserAuthenticator interface. This expands Derby's pluggable authentication logic to allow the customer to map authenticationIDs to custom Principals:

public Principal getPrincipal(String userName) throws SQLException;

I've been thinking more about this call.

Investigating how Principles are used, e.g. in the Subject class, a Subject (which is the object passed to doAsPrivileged) can contain more than one Principle. Thus to me it seems the getPrinciple() should have the flexiblity to return any number of Principles, so the return type should be a Set (see Subject) and thus can contain zero or more Principles. This then starts to get into the ability to support roles, e.g. an implementation could return three principles,
DatabasePrincipal("SYSTEM ADMIN"}
DatabasePrinciple("ACCOUNTING_DBA"}
X500Principal("cn=Alice")

and then this allows indirection between a specific user and a role.

Then thinking about implementing this interface I think the proposed api has two issues:

- if the authentication information is stored elsewhere (e.g. ldap) then getPrinciple() cannot work since no password is available
- it requires a double trip to the authentication store (e.g. two ldap calls)

Thus it seems to me the api should really be:

public Set authenticateUserWithPrinciples(String userName,
String userPassword,
String databaseName,
Properties info
)

or (I think I prefer this one)

// passing in a DatabasePrinciple, implementations can choose to included in the returned set or not.

public Set authenticateUser(DatabasePrinciple user,
String userPassword,
String databaseName,
Properties info
)

The maybe if the authenticateUser() that returns a Set returns null the existing old method is used.

Related to this, the functional spec does not specify what the Principle will be with the two builtin authentication schemes, BUILTIN and LDAP.
I assume for BUILTIN it will be DatabasePrinciple with the user's identifier, but what should it be with LDAP? Is there a standard Principle (e..g X500Principle) returned though the api we use to perform LDAP lookups?

35. **rhillegas:** Thanks for helping me puzzle through these issues, Dan. Here are some more thoughts triggered by your last comments:

    I think we need to be cautious as we start talking about roles. As I understand the javadoc, a Subject is a set of identities (that is, a set of Principals). A role, however, is a set of privileges--at least that's how ANSI models it. I think we will get into trouble if we mix these concepts together. That's one thought.

    Here's another: The idea that a person (a Subject) has multiple identities is very thought-provoking. Each of these identities may be granted permissions to perform work in some domain where that identity makes sense. Permissions are not granted to Subjects. They are granted to identities (Principals). So if you operate in multiple domains, you may need to be granted permissions as multiple Principals.

    I think that when a person connects to Derby (and passes authentication), they get a database identity, which is essentially their authorizationID. This database identity is in addition to whatever other identities they may have in the external authentication system. The external identities may be useful some day--for instance, the policy file may want to grant network permissions to these identities so that they can do work inside customer-written dbprocs. However, for our purposes today, it makes sense to me that Derby-specific permissions should be granted to database identities, that is, to DatabasePrincipals.

    So I'm wondering whether we need, right now, to beef up UserAuthenticator with a getPrincipal() or new authenticatUser() overload. We may need to do that in the future to help customers grant identity-conscious permissions to dbprocs. But it's not clear to me that we need this machinery now.

36. **djd:** I agree that supporting a more flexible Principle scheme could be added later, I don't think I see any issues with having a database login always having a DatabasePrinciple associated with it. I think for forwards compatibility that will be required, i.e. in the future methods in UserAuthenticator may be able to add Principles to the Subject, but will not be able to remove the DatabasePrinciple. I do think that Derby's internal implementation should not be relying on DatabasePrinciple, e.g. all fields, variables, parameters etc. are declared as Principle and DatabasePrinciple should only appear when the actual instance is created using new DatabsaePrinciple.

A few new thoughts did come out of your comment:

- with Java routines this spec does not indicate that the routine will run with the Subject set to one including the DatabasePrinciple. I don't think it needs to but it is a direction that may be required in the future. This would allow database user based granting on permissions in Java routines which might be useful since the only way to grant java permissions to code in installed jar files is to sign the jar and grant permissions to the signer.

- related to the last point, the visibility of the current Subject containing its DatabasePrinciple, might be clear to state that currently there is no visibility to user code of the Subject containing its DatabasePrinciple.

- Derby supports multiple databases within a system and per-database authentication. This means that ALICE in one database can be a different identity to ALICE in another database. DatabasePrinciple does not account for this. This may have the potential to be a security hole, though maybe not with the limited set of permissions today. Basic idea would be.

ALICE has some permissions that FRED does not have.
FRED only has permission to create databases
FRED can create a database with BUILTIN authentication and a user ALICE to pick up any permissions granted to ALICE.

It might be ok with the proposed changes because the identity is also authenticated in that specific domain, e.g. I assume to be ALICE to shutdown an engine or create a database one must be authenticated against the system authentication. Thus the ALICE from FRED's database would not be able to log into take advantage of the shutdown engine granted to the real ALICE.

The problems come when the policy file contains entries that are not linked to database authentication, e.g. to allow specific actions in routines, e.g. grant permission to ALICE to read '/etc/password', that could be hijacked by FRED's ALICE if the routine was run as the DatabasePrinciple(FRED).

I think this needs some thought, possible paths are: not an issue, DatabasePrinciple with database location path or good documentation the scope of DatabasePrinciple.

37. **rhillegas:** I agree that a DatabasePrincipal should encode both the database name and the authorization id inside that database. It is interesting that the same authorization id can have different credentials depending on the connected database.

    I don't know what the terms-of-art here are, but for the rest of this discussion, I'm going to use the following nomenclature:

    systemWideID - This is a user name that is authenticated with databaseName = null.

    databaseScopedID - This is a user name that is authenticated with a non-null databaseName.

    It is interesting that we authenticate the user twice when creating a database. First we authenticate with a systemWideID. If that succeeds, we create the database and mark that authorization id as the database owner. Then we re-authenticate the user as a databaseScopedID, using the same credentials. Clearly this assumes that at bootstrap time, the same credentials will work for the systemWideID and the databaseScopedID.

The policy file syntax for Principals is a little limited. That is, you're only allowed to declare one argument to your Principal's constructor. This means that we have to glue together the authorization id and database name. Maybe we can model this on the names used for KerberosPrincipal. Those names are of the form userName@realm. I don't know if the @ is going to be a nuisance. Any separator we choose will have escaping problems and @ may be particularly annoying to customers who want their authorization ids to be email addresses. But here's what it would look like:

this is a systemWideID
grant principal org.apache.derby.authentication.DatabasePrincipal "fred" ...

this is a databaseScopedID
grant principal org.apache.derby.authentication.DatabasePrincipal "fred@fredsDB" ...

this systemWideID is an email address
grant principal org.apache.derby.authentication.DatabasePrincipal "fred@@comcast.net" ...

this databaseScopedID is an email address
grant principal org.apache.derby.authentication.DatabasePrincipal "fred@@comcast.net@fredsDB"
...

I think that the create-database privilege should be granted to systemWideIDs for the following reasons:

1) The actual database creation today depends on whether we can authenticate the systemWideID, not the databaseScopedID.

2) This is a generic privilege which is not bound to a particular database name.

I think that the engine-shutdown privilege is also a systemWideID. So for this first release, I think we only need systemWideIDs--although the user guides should explain the implications of escaping @.

38. **fuzzylogic:** Unsetting Fix Version on unassigned issues.

39. **rhillegas:** Attaching 3rd rev of the functional spec. Thanks to everyone for sticking with this spec process and making this spec better. Your continued feedback is appreciated.

40. **fuzzylogic:** Hi Rick, thanks for the extra information regarding why Principals cannot use multiple string constructors in the policy file. I like the backslash escaping for principals included in the spec better than the others that were proposed earlier.
What is the expected behavior if a policy file has a grant block to a principal name that contains two unescaped @ symbols? Will these just be ignored?

41. **rhillegas:** Thanks for the quick feedback, Andrew. I think that two unescaped @ symbols should result in a syntax error. So, for instance, the following would be illegal:

DatabasePrincipal "foo@bar@wibble"

DatabasePrincipal "foo@@bar wibble"

Unless someone objects strongly, I will update the spec to say this.

42. **djd:** These two sentences jump out at me:

"The Release Notes and user guides will advise the customer that this should probably

be customized."
"Again the Release Notes and user guides will advise the customer to restrict these grants."

This is basically saying that we provide a "Basic policy" but we recommend against its use.
So this overall work is adding three policies, "basic", "custom" and "open", but the functional specifications are written to say one should use the custom policy and the other two are not recommended. That just seems strange, why go to all this bother since the custom policy can be done today, with no changes. :-)

Maybe we need to add a property for system authorization that specifies the default system administrator?

derby.system.defaultAdministrator

then the policy file could use DatabasePrinciple "${derby.system.defaultAdministrator}" rather than being wide open.
Maybe if derby.system.defaultAdministrator was not set it would default to "*". Another choice to to require the -u/-p arguments when starting the network server on the command line. Then the default user would be that matching -u.
Then the basic policy could be useful by itself.

Though the functional spec says:
"The Basic policy file grants this permission to everyone."
maybe the "everyone" is misleading here, as really its every user successfully authenticated against System authentication.

43. **rhillegas:** Attaching new rev of the functional spec, which makes these changs:

    1) renames the policy-reloading system procedure

    2) separates the Template policy from the Basic policy

44. **scotsmatrix:** Rick - This is a really detailed spec. Thank you for including the sections on Documentation and Release Notes.
    It is also especially useful to have the grant/revoke examples.
    The new name of the system procedure seems fine (from a doc standpoint :-) We try to limit names to 50 characters.

45. **rhillegas:** Hi Dan,

    Sorry for not responding to your January 26 comment sooner. I just noticed it.

    I think it would be fine to add a derby.system.defaultAdministrator property, default it to *, and grant the shutdown privilege to DatabasePrincipal "${derby.system.defaultAdministrator}".

46. **mzaun:** I've been looking into the System Privileges for a while and would like to publish some base classes required by this feature (my first derby patch) for review and discussion.

    While DatabasePrincipal and SystemPermission were more or less straight forward, there are a few finepoints with DatabasePermission (for instance, the use of canonicalized path names) which I've tried to document by comments inline.

SystemPrivilegesPermissionTest is an extensive unit test class with positive and negative tests cases on the features of the System- and DatabasePermission classes; this junit test uses a local policy file.

This suggested patch just provides the base classes and does not yet include the integration with the rest of derby.

47. **myrna:** I think this can be marked patch available - for review.

48. **rhillegas:** Thanks for this impressive patch, Martin. The Principal and Permissions classes look very polished to me. Hopefully, your next submission will be a little smaller: smaller patches are easier for the community to digest. I have committed this first increment, with the following changes, at subversion revision 544870:

    1) I commented out what looked like diagnostic scaffolding to me: a System.out.println() in DatabasePermission.implies(). If this was actually a useful piece of code, please explain.

    2) I commented out the running of the SystemPrivilegesPermissionTest in the junit _Suite. This is because the test has some problems, which I describe below and which you can address in a later submission.

    With these changes, the regression tests ran cleanly for me.

    Here are the problems which I saw in SystemPrivilegesPermissionTest:

    A) Right now, the test can only be run against the classtree, not against the jar files. This is because your Principal and Permissions classes are not pulled into derby.jar yet. They will be pulled into derby.jar when you actually reference these classes in the engine code. That is because the jar builder only includes classes that can be reached by following class references, starting at some distinguished entry points.

    B) When I ran this test standalone, I saw the following failure:

    .F
    Time: 0.425
    There was 1 failure:
    1) testSystemPrivileges(org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest)junit.framework.A
    expected IOException
    at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.checkDatabasePermission(SystemPrivil
    at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.execute(SystemPrivilegesPermissionTes
    at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.testSystemPrivileges(SystemPrivilegesP
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at org.apache.derbyTesting.junit.BaseTestCase.runBare(BaseTestCase.java:88)
    at junit.extensions.TestDecorator.basicRun(TestDecorator.java:24)
    at junit.extensions.TestSetup$1.protect(TestSetup.java:21)
    at junit.extensions.TestSetup.run(TestSetup.java:25)

    FAILURES!!!
    Tests run: 1, Failures: 1, Errors: 0

49. **dagw:** I saw some Javadoc warnings, e.g.

[javadoc] /export/home/tmp/derby/sb/sb1/java/engine/org/apache/derby/security/DatabasePermission.java:54: warning - Tag @see: missing '#': "DatabasePermission(String,String)"
etc.

50. **kristwaa:** 'derby-2109-03-javadoc-see-tags.diff' fixes some JavaDoc warnings (@see tags).

    Committed to trunk with revision 545514.

51. **kristwaa:** I just noticed that the 02 patch was uploaded without granting the license to ASF.
    Maybe that should be done?

52. **mzaun:** This patch adds support for the handling of special characters DatabasePrincipals names.

53. **mzaun:** Re-attaching the first patch with ASF license grant.

54. **mzaun:** Re-attaching DERBY-2109-04.diff (was incomplete).

55. **rhillegas:** Thanks for the patch, Martin. The system privilege test runs with the same error I was seeing before. You might want to sand down the error message in parsePrincipalName(): I think it should say that "name" rather than "action" shouldn't be null. Committed at subversion revision 546683.

56. **mzaun:** The attached document and patch is a 1st cut of the integration, and hence enforcement, of System Privileges. This patch has gotten somewhat larger that I'd initially anticipated and most likely needs further discussion.

    Please start with the document SystemPrivilegesTestCases.html, which expands on the Functional Specification and is meant for future inclusion into the spec. There's a bit of a deviation from the spec with regard to the case where no user credentials (username and password) are given by the user, but the server's Policy has a general grant for System Privileges, for instance, by a "wildcard" clause (DatabasePrinciple "*"). For this particular case allowing the requested operation (Engine Shutdown, Create Database) appears justified and has the advantage of backward compatibility. Note that explicit user credentials can be easily enforced by removing/restricting the "wildcard" grants in the server.policy. This is the implemented behaviour by the patch. Which has the benefit that the function tests, which generally don't provide credentials (for shutdown, for instance), pass -- due to the "wildcard" grant in the default server.policy. A stricter semantics that always required explicit user credentials (when running under a SecurityManager), would require changes to the junit test framework (provide user credentials in teardown(), for instance). However, this point about a given default identity may need further discussion.

    With regard, to design decisions made for this patch: There's a question where to put the checks that enforce System Privileges, whether to do them in services.monitor.BaseMonitor, for instance, or rather in the jdbc layer (InternalDriver, EmbedConnection). The embedded driver (as an outer layer) appears to be the better choice: there's a shutdown service, with which the checks should probably not interfere, and the right exception types are all there in the driver. In addition, NetworkServerControl needs to perform a privilege check before the shutdown URL is passed to the driver.

    Looking forward to your comments,
    Martin

57. **mzaun:** The attached patch DERBY-2109-06 addresses some issues with the unit test for System Privileges, described in Rick Hillegas's comments above: The unit test should now

run from the classes directory as well as the jar files and on all Operating Systems (there was a platform dependency, an attempt to provoke a non-canonicalizable filename error failed on Unix). In addition, a few unit test code cleanups.

Also, the unit test's policy file now contains all necessary grants.

Finally, junit.SecurityManagerSetup needed a small fix to refresh the Policies after a new SecurityManager has been installed (otherwise, the old Policies stay in effect).

Your comments appreciated,
Martin

58. **rhillegas:** Thanks for the patch and SystemPrivilegesTestCases.html, Martin. I will take a look at these in detail. Here's something that jumped out at me: could you break up the "no credentials" row into two rows like the other rows for credential settings? That would help me reason about the completeness of that table. Thanks!

59. **rhillegas:** Hi Martin,

    It looks like you've attached two patches, 05 and 06. It looks like 05 holds the product code and 06 holds the tests. I'm going to apply 05 first, then 06 and take the test for a drive. Does this sound right? Are these patches submitted for evaluation and discussion or are you asking that we commit them? Thanks.

60. **rhillegas:** Hi Martin,

    I'm having trouble applying the patches as I described. After I apply 05, I get the following error trying to apply 06:

    mainline (1.5) > svn_patch ~/junk/DERBY-2109-05.diffpatching file java/engine/org/apache/derby/impl/jdbc/EmbedConnection.java
    Reversed (or previously applied) patch detected! Assume -R? [n] C-c C-c

    Maybe you could supply a composite patch? I am going to wait for more guidance from you before I review these patches. Thanks.

61. **mzaun:**
    Hi Rick,
    I could not reproduce that patch issues you reported: I checked out a clean trunk, downloaded and applied patch 05 (patch -p0 < DERBY-2109-05.diff), then -06; no problems during patch and build. These two patches should be independent from each other, and a quick look didn't reveal any overlapping code to me.

    However, as a cross-check, I've attached a combined patch DERBY-2109-05and06.diff (from the freshly patched trunk) and hope this one works better.

    About the patches' status, yes, I consider patch:
    - DERBY-2109-05.diff, which contains the System Privileges enforcement code, ready for discussion (expecting questions), and
    - DERBY-2109-06.diff, which contains some System Privileges unit test fixes, ready for commital (or discussion if there are any questions). The unit test (suite) runs fine in my freshly patched and build trunk:
    java -cp "jars/sane/derbynet.jar;jars/sane/derby.jar;tools/java/junit.jar;jars/sane/derbyTesting.jar;" junit.textui.TestRunner org.apache.derbyTesting.unitTests.junit._Suite

If you find the unit test's two println()s unecessary, please, feel free to remove them:
testing System Privileges ...
testing System Privileges: done.

About the SystemPrivilegesTestCases.html document, I'll break out the "invalid credentials" case into two lines with a few updates (right after my return in a week).

Thanks for your comments so far, Martin

62. **knutanders:** Rick, it might just be a typo, but "svn_patch ~/junk/DERBY-2109-05.diff " as you mentioned above will reapply patch 05 instead of applying patch 06.

63. **rhillegas:** Thanks for the combined patch, Martin. This is really beautiful code showcasing some nice design patterns. I have a couple comments:

    1) EmbeddConnection and InternalDriver:

    The construction of the Subject is a bit involved and this code is duplicated for the createDB and shutdownEngine checks. I think it would be better if this construction of the Subject were abstracted out into a shareable piece of code.

    2) NetworkServerControlImpl:

    I think that there is no need to invent the concept of a default password for the APP account. I think it would be better to default passwordArg to "" or null.

    init() and checkShutdownPrivileges()

    I notice that init() always sets the user and password properties but checkShutdownPrivileges() only sets them if the are not null. What is the reason for this asymmetry?

    3) server.policy

    I would expand the comment to punch up the message that these are broad privileges which we encourage the user to narrow. Also, you will want to make corresponding changes to java/drda/org/apache/derby/drda/template.policy

    4) derby_tests.policy

    As a bit of defensive coding, you may want to make corresponding changes to the other policy files used by the regression tests.

    5) SystemPrivilegesPermissionTest

    I would convert the System.out.println() diagnostics to println() calls. That way you will see this output when you turn on debugging, but the printout won't show up for ordinary regression test runs.

    6) I think that 05 and 06 must be committed together. 05 wires in the references which are necessary to pull the new permission and principal classes into the jar files. Without these references, the

new regression tests will fail if run against jar files.

64. **mzaun:** The attached DERBY-2109-07 patch replaces DERBY-2109-05and06.

    I addressed all of Rick's comments, except for 2) (default user identity for NetworkServer-Control), which turned out to be non-trivial. I'll add a detailed comment specifically on this issue along with an update to SystemPrivilegesTestCases.html.

    Also, the patch contains fixes to the test suites and now passes junit-all and derbyall on both the derby jar files and the classes directory.

    Please, review, test, and comment.

65. **mzaun:** Reattached DERBY-2109-07 files with ASF inclusion grant.

66. **rhillegas:** Hi Martin,

    Thanks for the even better next rev of the patch. Once again, the code is beautiful. I have a couple comments:

    1) EmbedConnection: It looks to me as though the CREATE DATABASE
    privilege will not be enforced if the user is creating the database
    from a backup copy using the "createFrom" URL attribute. I think this
    is fine and the functional spec doesn't address this issue. As an
    add-on effort we may want to consider if we want to restrict the
    ability to create a database from a backup copy. This may be a new
    database privilege ("createFrom" instead of "create") or it may be the
    same privilege. In any event, the topic is worth some thought
    regardless of whether we address it in 10.4.

    2) SecurityUtil: There seems to be a fair amount of code shared
    between checkShutdownPermission() and
    checkDatabaseCreatePermission(). In a small, future patch there's an
    opportunity to factor this shared code into a common routine, say,

    checkUserHasPermission(String user, Permission permission)

    3) NetworkServerControlImpl:

    a) I notice at line 858 that user and
    password properties are unconditionally stuffed into a Properties
    object without checking whether they are null. In contrast, at line 1026
    these properties (in other Properties objects) are guarded by null
    checking. I think that a NullPointerException will be raised if the
    values are null. Are we sure that the username and password will
    always be non-null at line 858?

    b) processCommands(): I notice that the protocol includes
    optional username/password arguments now. I just want to confirm
    that the protocol won't do something silly if username or password
    is null or if password is specified but username isn't.

67. **rhillegas:** After applying this patch, the regression tests run cleanly for me except for a diff in outerjoin.sql. That regression was not introduced by this patch--I also see the diff in a clean client newly cut from the mainline:

    MasterFileName = master/outerjoin.out
    1737 del
    < Empty right rows returned = 1
    1737a1737
    > Empty right rows returned = 0
    Test Failed.
    *** End: outerjoin jdk1.5.0_07 2007-12-07 13:26:01 ***

68. **rhillegas:** The regression tests also pass for me when run against the classpath rather than the jars--modulo the outerjoin.sql problem noted above.

69. **mzaun:** Rick,
    thanks for your comments, I've incorporated 1), 2), and 3.a) and verified 3.b) and attached a new patch DERBY-2109-08 replacing DERBY-2109-07. I'd very much appreciate if you could give some scrutiny to the new document SystemPrivilegesBehaviour.html (replacing deleted SystemPrivilegesTestCases.html).

    Developers,
    with the latest patch DERBY-2109-08, I consider the work on System Privileges becoming ready for integration.

    By the specification of this feature, there will be some incompatibilities once integrated. For instance, users will have to provide credentials to "NetworkServerCommand shutdown" when running with authentication. Users with customized server.policy files will have to add a couple of permissions (when running with Authentication and SecurityManager).

    The user-visible changes with System Privileges and the error messages in case of failures are summarized and described by attached document SystemPrivilegesBehaviour.html. Please, have a close look and provide feedback.

    Thanks! Martin

70. **rhillegas:** Hi, Martin. Thanks for the patch and the improved description of the compatibility issues. I am running the tests now.

    I don't want to commit this work until the community agrees on how we want to handle the compatibility issues you have noted. I believe that we will need a community discussion and a release note.

    Before starting a community discussion, I would like to make sure that I understand what the non-backward-compatible cases are. This is my understanding. Please correct me as necessary.

    The non-backward-compatible cases arise for customers who do BOTH of the following:

    A) Run with Authorization turned on

B) Run with a Java Security Manager

For these customers, the non-backward-compatible cases are:

1) If the customer has written their own policy file, then the customer will need to add some more permissions to it. These permissions are needed in order to create databases and shutdown the engine.

2) If the customer shuts down the network server via NetworkServerControl, then the customer will need to supply credentials to the shutdown command.

Does this sound right? If not, could you summarize the situation better?

Thanks,
-Rick

71. **djd:** Can the specification clarify what SystemPermission "shutdownEngine" gives permission to do?

    From the name I would think it controls the permission to shutdown the Derby embedded database engine, but later in the spec it says shutting down the network server requires this permission. Is that just because shutting down the network server will shutdown the engine?

    I thought that shutting down the network server and database engine were independent operations, thus shouldn't there be two permissions, shutdownEngine and shutdownServer?

72. **rhillegas:** Hi Dan. The spec describes only one shutdown permission, shutdownEngine. If you have this privilege, then you can shutdown the engine and you can shutdown the network server too.

    As a follow-on patch or effort, we could add a separate shutdownServer permission. If we did this, then I think that it would make sense that shutdownServer => shutdownEngine. I am unable to think of a reason that one would want someone to have the ability to shutdown the VM but not the engine. At first blush, it ought to be possible to implement this relationship via Permission.implies().

    The following cases arise:

    1) Neither permission is granted. Neither the server nor the engine can be brought down gracefully.

    2) Only shutdownEngine is granted. The engine can be brought down gracefully but the server cannot be.

    3) Only shutdownServer is granted. Both the engine and the server can be brought down gracefully.

    4) Both shutdownServer and shutdownEngine are granted. Both the engine and the server can be brought down gracefully.

    (1) and (2) seem like mistakes to me. (3) and (4) look very similar to one another.

    Creating a separate shutdownServer permission allows one to have a user who enjoys the

permission to shutdown the engine but not the server. I would like to understand the difference between the ServerAdministrator and EngineAdmistrator roles. What use-cases are supported by the additional role?

Here are some approaches which we could take in follow-on patches and efforts:

1) Rename the shutdownEngine permission to just shutdown. That would correspond better with the behavior described in the spec and implemented in this patch.

2) Either in 10.4 or a later release, implement a separate shutdownEngine permission if the use-cases seem compelling. The behavior would be shutdown => shutdownEngine.

73. **mzaun:**
Rick,
thanks for the summary on the backward compatibility issues, which only arise, as you've correctly described, under both, A) and B).
For 100% completeness, I'd have two technical additions:

> Rick Hillegas wrote:
>
> The non-backward-compatible cases arise for customers who do BOTH of the following:
>
> A) Run with Authorization turned on
>
> B) Run with a Java Security Manager
>
> For these customers, the non-backward-compatible cases are:
>
> 1) If the customer has written their own policy file, then the customer will need to add some more permissions to it. These permissions are needed in order to create databases and shutdown the engine.

...and to run the security checks themselves (which require an AuthPermission, a PropertyPermission, and a FilePermission -- see template.policy).

>
> 2) If the customer shuts down the network server via NetworkServerControl, then the customer will need to supply credentials to the shutdown command.
>

... and
3) If the customer shuts down the engine via a connection URL with "shutdown=true;", then the customer will also need to supply credentials, i.e., user/password attributes in the URL.

> Does this sound right? If not, could you summarize the situation better?
Good summary. Thanks, Martin

74. **rhillegas:** Thanks Martin,

Just to clarify, the additional case (3) is true today isn't it? Today if Authorization is turned on, then you must pass valid credentials in order to shutdown the engine via an URL? So (3) would not be an additional backward-compatibility case, just a continuation of current behavior?

Thanks,
-Rick

75. **mzaun:**
Rick,

>Just to clarify, the additional case (3) is true today isn't it? Today if Authorization is turned on, then you must pass valid credentials in order to shutdown the engine via an URL? So (3) would not be an additional backward-compatibility case, just a continuation of current behavior?
"Yes" to all three questions -- I overlooked that 3) is not newly introduced but already there.

Thanks, Martin

76. **djd:** For the shutdown issue I thinking of the situation where the application is using the embedded mode but supports an optional client/server mode for debugging/reporting etc. In that case a user may want to start and stop the server without stopping the embedded engine. Thus they may have permission to stop the server but not to stop the engine.

77. **djd:** » 1) If the customer has written their own policy file, then the customer will need to add some more permissions to it. These permissions are needed in order to create databases and shutdown the engine.

> ...and to run the security checks themselves

What does "run the security checks themselves" mean here?

78. **mzaun:**
I just noticed that a few diffs to 4 derbyall "master output" files were missing from DERBY-2109-08. They reflect the changed "usage" message printed by NetworkServerControl for "shutdown" (new user/password arguments).

Please find attached DERBY-2109-08_addendum, which complements DERBY-2109-08. With the additional patch, entire derbyall passes in my workspace.

Martin

79. **rhillegas:** Hi Dan,

Martin attached a document called SystemPrivilegesBehaviour.html. In that document in the section titled "Policy File", there are two blocks of permissions. The first block of permission are needed in order to run the doAsPrivileged() method (that is, to run as a particular DatabasePrincipal) and to canonicalize file names on behalf of the createDatabase checks. I believe that this is what Martin means by "run the security checks themselves".

80. **mzaun:**
> I believe that this is what Martin means by "run the security checks themselves".

Correct. The code implementing the autherization checks needs to
- run a doAsPrivileged (hence the javax.security.auth.AuthPermission)
- resolve relative path names (hence the java.util.PropertyPermission), and
- canonicalize path names (hence the java.io.FilePermission).

These permissions are only needed on derby.jar. See SystemPrivilegesBehaviour.html.

Thanks, Martin

81. **djd:** > > I believe that this is what Martin means by "run the security checks themselves".

    > Correct. The code implementing the autherization checks needs to
    Just to confirm, "the code" here means Derby code, not code written by the user, right?
    It's just the "themselves" seems to imply the user now has to write code to implement Derby security checks.

82. **mzaun:** >Just to confirm, "the code" here means Derby code, not code written by the user, right?

    Correct.

    > It's just the "themselves" seems to imply the user now has to write code to implement Derby security checks.

    My language wasn't clear.

    Martin

83. **johnemb:** This is confusing. Are you (Rick/Martin) mixing authentication with authorization in the most recent comments to this issue? If not, please help me understand what kind of Authorization we are talking about here:

    --- --- ---
    > The non-backward-compatible cases arise for customers who do BOTH of the following:
    » A) Run with Authorization turned on
    » B) Run with a Java Security Manager
    --- --- ---
    » Just to clarify, the additional case (3) is true today isn't it? Today if Authorization is turned on, then you must pass valid credentials in order to shutdown the engine via an URL? So (3) would not be an additional backward-compatibility case, just a continuation of current behavior?» "Yes" to all three questions -- I overlooked that 3) is not newly introduced but already there.
    --- --- ---

    I think that the case mentioned above is true if authentication is turned on, regardless of any settings for authorization.

    Regarding the SystemPrivilegesBehaviour.html document: I find it very useful in understanding the implications of these changes, especially the table with all the combinations of configurations. I find the second bullet under the "Changes" heading a bit unclear, though:

    --- ---
    (...users must) * have autherization by the used Java Policy for engine shutdown and/or create database.
    --- ---

    I think I know what is meant, but please correct me if I'm wrong:
    (users must) be authorized by the used Java Security Policy to perform engine shutdown and/or create a database.

84. **rhillegas:** Hi John,

    >This is confusing. Are you (Rick/Martin) mixing authentication with authorization in the most recent comments to this issue? If not, please help me understand what kind of Authorization we are talking >about here:

    You are right. When I wrote above on January 11 "A) Run with Authorization turned on ", I meant "Run with Authentication turned on". Martin describes the situation correctly in SystemPrivilegesBehaviour.html.

    >I think I know what is meant, but please correct me if I'm wrong:
    >(users must) be authorized by the used Java Security Policy to perform engine shutdown and/or create a database.

    Yes. If you are BOTH running with Authentication turned on and with a Java Security manager, then you need to be authorized by the security policy to shutdown the engine and create databases.

85. **rhillegas:** I have run the regression tests against DERBY-2109-08 with its addendum. The tests ran cleanly for me.

86. **djd:** With the class DatabasePrincipal, a user name of "*" corresponds to all users. Is this use of * come from any existing practice? In SQL authorization the identifier PUBLIC is used to represent all users. Would it make more sense to use the SQL practice here?

    Given that the representation of user identifiers can cause confusion (see http://wiki.apache.org/db-derby/UserIdentifiers) it would be good if the DatabasePrincipal javadoc and the functional spec indicated how it (with examples) handles user name (in the code and the policy file). It looks like user names would be entered in their normal form in the policy file unless they include one of the special characters *, \ and @. What about if the user name includes a double quote?

    It's also worth noting that policy file and the DatabasePrincipal are using back-slash as an escape, thus a normalized user name of eve* would have to eve* in the policy file (I think). A single back-slash in the user name would be four backslashes in the policy file.

    In DatabasePrincipal at line 80 there is a comment that the "general rule" is to have english only messages for "internal coding errors". First - where does this general rule come from, I've never heard of it for Derby.
    Second - many of the english only messages are not internal coding errors, but configuration errors in the policy file.

    Also several of the messages in DatabsaePrinicpal refer to "action" when I think they mean name.

87. **djd:** I think the patch has some issues with name handling:

    EmbedConnection takes the user name from the connection request and passes it into SecurityUtil.checkDatabaseCreatePermission.
    That method takes the name and passes it into the constructor for DatabasePrincipal.

    The problem is that the rules for the format of the name in DatabasePrincipal do not match the format of the name for connection requests, this is due to the special escaping

that happens in DatabasePrincipal to cope with the special characters *, @ and \.

88. **rhillegas:** Hi Dan,

    >With the class DatabasePrincipal, a user name of "*" corresponds to all users. Is this use of * come from any existing practice? In SQL authorization the identifier PUBLIC is used to represent all users. >Would it make more sense to use the SQL practice here?
    I'm not sure that we can use PUBLIC as a username wildcard. My cursory reading of the SQL standard suggests that any delimited identifier can be a valid user name and authorization id. This includes "PUBLIC", "TABLE", "GRANT", etc.. I can see that Derby treats the delimited identifier "PUBLIC" as equivalent to the keyword PUBLIC in GRANT statements--but is this correct?

    Can you point me at a clause in the spec which forbids the use of "PUBLIC" as a user name?

89. **djd:** 5.4 SR 20) No <authorization identifier> shall specify "PUBLIC".

90. **johnemb:** Dan commented:
    > With the class DatabasePrincipal, a user name of "*" corresponds to all users. Is this use of * come from any existing practice? In SQL authorization the identifier PUBLIC is used to represent all users. Would it make more sense to use the SQL practice here?

    At first glance I found it intuitive that the wildcard for "all principal names" is "*", since a number of Permissions in a Java Security Policy file already accept * as a wildcard (e.g. for host names, property names, file paths, etc.). I don't know much about existing practice with regards to principal wildcards, but I found this [1]:

    "The principal_class_name may be set to the wildcard value, *, which allows it to match any Principal class. In addition, the principal_name may also be set to the wildcard value, *, allowing it to match any Principal name. When setting the principal_class_name or principal_name to *, do not surround the * with quotes. Also, if you specify a wildcard principal class, you must also specify a wildcard principal name."

    [1]: http://download.java.net/jdk7/docs/technotes/guides/security/PolicyFiles.html#FileSyntax

    So, I tried using a customized policy file with the latest patch, and from what I could see, using
    grant principal org.apache.derby.authentication.DatabasePrincipal * {

    is, from a user's perspective, equivalent to

    grant principal org.apache.derby.authentication.DatabasePrincipal "*" {

    So, even if the wildcard in our implementation is changed to "PUBLIC" or something else, it seems that users can still use the generic wildcard * (no quotes) to specify "all users" (correct me if I'm wrong).

    Other than that, I agree with Dan's comment about the need to specify how the various forms of user names are handled (preferably in the user documentation as well, not only in the funcSpec). The current handling/presentation/usage of user names in Derby is IMHO already quite confusing, if not a mess, so it would be good not to add too many extra variables into the mix.

Finally, I would like to mention that I have done some (basic) manual experiments using Derby with the latest patches for this issue applied, and have found no issues so far.

91. **djd:** Thanks John, but is that wildcard value specific to Java 7? The only text I can find in J2SE 5 and JDK 6 is:
- The principal field is optional in that, if it is omitted, it signifies "any principals".

Either way, it seems there are already mechanisms to specify all users, thus we don't want to introduce another way of specifying all users by having a quoted *.
Supporting PUBLIC as a DatabasePrincal name to represent all users could be a follow on change. It might make it clearer for folks used to SQL authorization.

92. **johnemb:** I have found the wildcard value of * for principal_name mentioned only in the JDK7 docs, but I tried it on Sun's JVMs 1.4.2 and 1.5.0, and it worked the same way, as described. Not sure if we can/should rely on that or not, and I don't know if other vendors do the same thing.

93. **djd:** Rick wrote:
> 1) Rename the shutdownEngine permission to just shutdown. That would correspond better with the behavior described in the spec and implemented in this patch.
> 2) Either in 10.4 or a later release, implement a separate shutdownEngine permission if the use-cases seem compelling. The behavior would be shutdown => shutdownEngine.
I think these are good ideas, namely rename the permission to shutdown and then optionally later add in more specific shutdown permissions.

94. **mzaun:** Attached is an upated version of the SystemPrivilegesBehaviour.html extending the description of the user-visible changes and the backward compatibility issues, which was incomplete.

By the functional spec and the implementation, this JIRA addresses two issues, which could have been separated:
a) authentication: extend NetworkServerControl to support user credentials
b) authorization: introduce checks for EngineShutdown/CreateDatabase System Privileges

The reason it came about is because b) requires a) (but not vice versa).

The backward compatibility issues with a) and b) are separate as now highlighted in the SystemPrivilegesBehaviour document.
(Also, see the "Compatibility issue for 10.4" discussion on derby-dev.)

Martin

95. **mzaun:**
Hi Dan,
thanks for the pointer and your comments on the rules for User Identifiers.

> Daniel John Debrunner wrote:
> Given that the representation of user identifiers can cause confusion (see http://wiki.apache.org/db-derby/UserIdentifiers) it would be good if the DatabasePrincipal javadoc and the functional spec indicated how it (with examples) handles user name (in the code and the policy file).
I see: we need to document some extra rules for user names in policy files, because
a) we're bound to the policy file syntax, i.e., the name argument to DatabasePrincipal needs to be a Java String literal in double quotes (in contrast to the use of identifiers in SQL) and
b) the functional spec introduces the special characters *, @, and \, which otherwise are

ordinary characters in user identifiers.

So, I agree it's of importance to document this in the javadoc, functional spec, the above wiki page, and also as comments in server.policy and template.policy.

I'll update my patch (javadocs, policy files) with more comments and can also add some examples to the User Identifiers wiki page when we have agreement.

Before replying to your detailed comments, let me try to backtrack and summarize the cases:

1) Non-delimited user identifiers:
1.1) I understand the User Identifier document as saying that identifiers are case-INsensitive unless delimited by double-quotes.
1.2) The policy file syntax requires the argument to DatabasePrincipal to be a Java String literal, i.e., enclosed in double-quotes.
1.3) The enclosing double-quotes imposed by the policy file syntax shouldn't count towards the user name, since they're mandated by Java (consistent with the examples in the User Identifiers document), e.g., DatabasePrincipal "eve" displays a non-delimited identifier.
1.4) According to the rules, non-delimited user names in policy files may be written all-uppper, -lower, or mixed case but are to be interpreted case-INsensitively, e.g., DatabasePrincipal "eVe" and DatabasePrincipal "EVE" both mean the same for authentication and authorization purposes,1.5) The current patch does not yet perform the authorization checks for non-delimited user names on their normal form -- but it shouldn't be too difficult to add this once we have agreement.

2) Delimited user identifiers:
2.1) Users can form a delimited identifier in policy files by putting escaped double-quotes into the String literal (consistent with the examples in the User Identifiers document), e.g., DatabasePrincipal "\"eVe\"" displays a delimited identifier.
2.2) Those delimited user names are to be interpreted case-sensitively for authentication and authorization purposes.
2.3) I've tested that the DERBY-2109 patch works with delimited identifiers in policy files and authenticated user names and handles them case-sensitively. So, all seems to be done here.

3) User identifiers with special System Privileges characters *, @, or \:
3.1) According to the functional spec, these special characters need to be escaped when they're just part of a DatabasePrincipal user name.
3.2) By the Java syntax, the backslash character itself needs to be escaped, e.g., a grant for user M@rtin reads DatabasePrincipal "M\\@rtin".
3.3) This means that within the DERBY-2109 autherization checks, all special characters in authenticated user names need to be escaped by \ before they're matched against the DatabasePrincipal.
3.4) The published patch does not yet apply that escaping of special characters, as you've rightfully pointed out, but I've successfully tested a quick fix already (see below).

> It looks like user names would be entered in their normal form in the policy file unless they include one of the special characters *, \ and @.
I don't follow: What's the issue with special characters in normal forms?
For example, we should be able to support all of these:

a) all feasible, all denote the same user:
DatabasePrincipal "M\\@rtin"
DatabasePrincipal "m\\@rtin"

DatabasePrincipal "M\\@RTIN"

b) all feasible, but each denotes a different user:
DatabasePrincipal "\"M\\@rtin\""
DatabasePrincipal "\"m\\@rtin\""
DatabasePrincipal "\"M\\@RTIN\""

> What about if the user name includes a double quote?

Works, see 2.3) above.

> It's also worth noting that policy file and the DatabasePrincipal are using back-slash as an escape, thus a normalized user name of eve* would have to eve* in the policy file (I think).

Correct. Also see 3.2) above.

> A single back-slash in the user name would be four backslashes in the policy file.

Correct. For instance, user M\rtin would receive a policy grant as DatabasePrincipal "M\\\\rtin"

> I think the patch has some issues with name handling:
> EmbedConnection takes the user name from the connection request and passes it into SecurityUtil.checkDatabaseCreatePermission.
> That method takes the name and passes it into the constructor for DatabasePrincipal.
> The problem is that the rules for the format of the name in DatabasePrincipal do not match the format of the name for connection requests, this is due to the special escaping that happens in DatabasePrincipal to cope with the special characters *, @ and \.
Good catch. I've tested a quick fix, see 3.4) above, which I'll publish soon.

The best place to perform the escaping of special System Privileges characters is within the method
org.apache.derby.security.SecurityUtil.createDatabasePrincipalsSubject(String user)
right before the DatabasePricipal is instantiated with the user name. That way, the escaping is done for all authorization checks (Rick, thanks for having suggested that small code refactorization!) and the modified user name stays local.

Martin

96. **mzaun:**
> John H. Embretsen wrote:
> I have found the wildcard value of * for principal_name mentioned only in the JDK7 docs, but I tried it on Sun's JVMs 1.4.2 and 1.5.0, and it worked the same way, as described. Not sure if we can/should rely on that or not, and I don't know if other vendors do the same thing.

The javadocs of a 1.4 Sun JDK class
http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/PolicyFile.html
has a paragraph explicitly describing the use of * (recommended without double-quotes) as wildcard for Principal names and classes (but that class has been deprecated for other reasons).

But the general concept of "*" as wildcard character in policy files is documented:

http://java.sun.com/j2se/1.5.0/docs/guide/security/PolicyFiles.html#Examples

> Daniel John Debrunner wrote:
> Either way, it seems there are already mechanisms to specify all users, thus we don't want to introduce another way of specifying all users by having a quoted *.
> Supporting PUBLIC as a DatabasePrincal name to represent all users could be a follow on change. It might make it clearer for folks used to SQL authorization.

One can argue that with System Privileges (server shutdown, create database) we're in the realm of Java Security, not SQL authorization, controlled by Java policy (file) permissions in contrast to grants/revokes.

If we feel that we're at the intersection of SQL and Java Security and that we want to please users with either background -- we can easily support both, PUBLIC and * as wildcard characters. Of course, allowing both rather adds to the documentation.

97. **djd:** With the format of user name for DatabasePrincipal we are heading towards a non-standard format of user name and one that becomes hard to read with delimited identifiers. I wonder if a better approach would be to use a standard here, namely X500 and have DatabasePrincipal extend javax.security.auth.x500.X500Principal.

    The UID component could represent the Derby user name and in the future DC (domain component) could represent the database.

98. **djd:** > 1) Non-delimited user identifiers:
    > 1.1) I understand the User Identifier document as saying that identifiers are case-INsensitive unless delimited by double-quotes.

    This is where it gets somewhat complicated, that's not a blanket true statement. A user name can be submitted to Derby in a number of contexts, in some of those contexts unquoted input is case-insensitive, in others it is not.

    Examples of case insensitive:
    Regular SQL identifiers
    user property in a JDBC connection request

    Examples of case sensitive:
    SQL value passed to a procedure expecting a user name
    SQL value representing a user name in a system table

    I was originally assuming that a user name in a policy file would be case-sensitive, thus
    grant DatabasePrincipal "eve", DatabasePrinicpal "EVE"
    would grant the permissions to two different users.

    However that may not be consistent with the usual way to input user names to Derby, which seems on the whole to use the rules for SQL identifiers.

99. **rhillegas:** I would like to share some thoughts about the case-sensitivity of DatabasePrincipals. It seems to me that there are at least three concepts of identity in play here:

    1) UserName - This is what is passed to the authentication service as part of credentials checking.

2) AuthorizationID - This is the owner of a schema and the grantor and grantee of fine-grained SQL privileges via the GRANT/REVOKE commands

3) DatabasePrincipal - This is the recipient of shutdown and createDatabase privileges

UserName and AuthorizationID are not the same thing. Unfortunately, the spec attached to this issue says that they are the same thing. That's just a mistake and it needs to be cleaned up.

To illustrate the difference between UserNames and AuthorizationIDs, consider the following: From the point of view of the authentication service, the following are all distinct UserNames:

Edward
EdWard
EDWARD
"Edward"
"EdWard"

These, however, map to only 3 distinct AuthorizationIDs:

Edward = EdWard = EDWARD
"Edward"
"EdWard"

The practical consequence of this is that Edward and EdWard will authenticate with different credentials but will be thrown into the same schema and will be able to view and edit one another's data. That is, the following two connection URLs authenticate with different credentials but are thrown into the same schema and are treated by GRANT/REVOKE as the same person:

connect 'jdbc:derby:mydb;user=Edward;password=EdwardPassword';
connect 'jdbc:derby:mydb;user=EdWard;password=EdWardPassword';

Derby's solution to this problem is to tell the customer that they need to add two new users to their authentication system: "Edward" and "EdWard" so that Derby can disambiguate these users. This is frustrating to customers who want to integrate Derby applications into company-wide processes which rely on a single, organization-wide authentication scheme.

I suspect that this behavior goes back to the introduction of CREATE SCHEMA long before Derby was open-sourced.

I think it would be unfortunate if we added yet another concept of identity. So it makes sense to me that a DatabasePrincipal represents either a UserName or an AuthorizationID. But which one?

AuthorizationID seems like a good fit once you are operating inside a database. I have reservations about using it at the system-wide level where there are no schemas or GRANT/REVOKE privileges.

I also have misgivings about the following scenario:

A) There is a system administrator with UserName Edward.

B) Many applications run in the system, including a Payroll application which has a user named EdWard.

C) I want to grant shutdown privilege to Edward but not to EdWard.

I am wondering whether we should abandon the idea of using the policy file to control database-specific privileges (other than createDatabase itself). This would mean that in the future we plan to use GRANT/REVOKE extensions to manage privileges once you are inside a database. The policy file would then only be used for system-wide privileges where you do not have a database context. If we went down this path, then we would get rid of the @ syntax and many of the escaping cases which are proving to be so confusing. In this world, a DatabasePrincipal would correspond to a system-wide UserName.

100. **djd:** Rick - good points with the clarification of user name & authentication id, though for this:

> The practical consequence of this is that Edward and EdWard will authenticate with different credentials

I think this is a bug in the BUILTIN authentication scheme, not a design of the authentication system. The documentation for UserAuthenticator.authenticateUser kind of states this, but could be clearer. It applies case sensitivity to authorization identifiers when it really means user names.

> Derby's solution to this problem is to tell the customer that they need to add two new users to their authentication system: "Edward" and "EdWard" so that Derby can disambiguate these users. This is frustrating to customers who want to integrate Derby applications into company-wide processes which rely on a single, organization-wide authentication scheme.
I think that's one solution, but not the most optimal. The best solution would be to implement your own UserAuthenicator which translates from the user name format to the format expected by the company wide authentication scheme. Ideally Derby's LDAP authentication would have a builtin option (if not the default) to do this, e.g. always authenticate against the LDAP server using the authorization identifier.

> In this world, a DatabasePrincipal would correspond to a system-wide UserName.

and then it would make sense to rename the class to be SystemPrincipal.

101. **rhillegas:** The wildcarded Principal name is only needed for the case that we want to grant privileges to principals of the form *@databaseName (or PUBLIC@databaseName). If we agree that we aren't going to use the policy file to regulate privileges at the database level, then the existing policy file syntax will cover the remaining wildcard case (the system-wide wildcard) as John and Martin have pointed out:

grant {
// grant any user to shutdown the engine
permission org.apache.derby.security.SystemPermission "shutdownEngine";

// grant any user to create databases anywhere
// (to succeed actual locations require additional file permission grants)
permission org.apache.derby.security.DatabasePermission "directory:«ALL FILES»", "create";
};

By eliminating the two special characters, * and @, we eliminate most of the escaping cases. And we don't need to specify PUBLIC in the policy file. If SystemPrincipal represents a system-wide UserName, then I think we clear up a lot of the confusion.

102. **rhillegas:** Based on the recent discussions, I propose to change the functional spec as follows--unless there are strong objections:

    1) Change the "shutdownEngine" privilege to be just "shutdown". This is a better match for the behavior of this privilege, which controls the ability to shutdown both the server and the engine. In the future, someone can add a shutdownEngine privilege with the behavior that shutdown => shutdownEngine.

    2) Change the name of the Principal class from DatabasePrincipal to SystemPrincipal. This class will describe system-wide Principals. This eliminates the need for the special @ syntax. Wildcarding can be accomplished as John and Martin have described, so there is no need for the special * or PUBLIC syntax. In the future, we can introduce a DatabasePrincipal if use cases for this arise. We introduce no additional escaping cases on top of the escaping syntax defined for the policy file.

    3) Clarify that a SystemPrincipal represents a UserName. It is therefore case-sensitive.

103. **djd:** > 3) Clarify that a SystemPrincipal represents a UserName. It is therefore case-sensitive.

    Do you mean authorization identifier instead of UserName since it's authorization identifiers that are case sensitive?

    Or did you mean "follows the rules for SQL identifiers" instead of "case-sensitive", since some user names are folded to upper case (and therefore appear case-insensitive) and some are stripped of their delimiting quotes (and hence appear case-sensitive). Ie. the rules are more compilcated than just case sensitivity.

104. **rhillegas:** » 3) Clarify that a SystemPrincipal represents a UserName. It is therefore case-sensitive.
    >
    >Do you mean authorization identifier instead of UserName since it's authorization identifiers that are case sensitive?>
    >
    >Or did you mean "follows the rules for SQL identifiers" instead of "case-sensitive", since some user names are folded to upper case (and therefore appear case-insensitive) and some are stripped of their >delimiting quotes (and hence appear case-sensitive). Ie. the rules are more compilcated than just case sensitivity.

    Here I'm following up on my musings about the three concepts of identity which I see in play here:

    1) UserName -- this is part of the credentials passed to the authentication service. This could be case-sensitive or case-insensitive depending on the rules of the authentication service.

    2) AuthorizationID -- this is the SQL concept of identity. This is case-insensitive unless double-quoted.

    3) SystemPrincipal -- I think that we would like to map this onto either (1) or (2). I

think that (1) is a better fit than (2). At the system level there is no SQL context and the behavior/sobjects being controlled are not SQL behaviors/objects.

105. **rhillegas:** Just to clarify that username/password credentials are case-sensitive, I ran the following experiment: I created my own UserAuthenticator implementation, wired it into derby.properties, and then connected. The custom UserAuthenticator printed out the arguments which Derby passed to it. Those arguments are case sensitive, so it is up to the UserAuthenticator to determine the rules it wants to enforce.

    Here is my custom UserAuthenticator:

    ```
    import org.apache.derby.authentication.UserAuthenticator;

    import java.util.Properties;
    import java.sql.SQLException;

    /**
    * Dummy authenticator. All users are legal
    *
    * @see org.apache.derby.authentication.UserAuthenticator */

    public class DummyAuthenticator implements UserAuthenticator
    {
    public DummyAuthenticator() {}

    /**
    * Authenticate the passed-in user's credentials.
    * A more complex class could make calls
    * to any external users directory.
    *
    * @param userName The user's name
    * @param userPassword The user's password * @param databaseName The database *
    @param infoAdditional jdbc connection info.
    * @exception SQLException on failure
    */
    public boolean authenticateUser(String userName,
    String userPassword,
    String databaseName,
    Properties info)
    throws SQLException {
    System.out.println( "userName = " + userName + "\nuserPassword = " + userPassword +
    "\ndatabaseName = " + databaseName );
    System.out.println( "info = " + info );

    return true;
    }
    }
    ```

    Here is my derby.properties:

    ```
    derby.connection.requireAuthentication=true
    derby.authentication.provider=DummyAuthenticator
    ```

And here is an ij script showing how this behaves at run-time:

```
ij version 10.4
ij> connect 'jdbc:derby:derby10.4;user=myHumbleSelf;password=myHumblePassword';
userName = myHumbleSelf
userPassword = myHumblePassword
databaseName = derby10.4
info = {user=myHumbleSelf, password=myHumblePassword}
ij> select count(*) from sys.systables;
1 -----------
24
1 row selected
ij> userName = null
userPassword = null
databaseName = null
info = {shutdown=true}
```

106. **djd:** Rick wrote:

> 2) AuthorizationID -- this is the SQL concept of identity. This is case-insensitive unless double-quoted.
> We seem to be using the same terms for slightly different concepts.

I believe I'm using authorization identifier in the way that the SQL standard defines it. This would be a case-sensitive value that defines a unique identity in the sql system (database). Thus an authorization identifier is never double quoted, contradicting your statement 2).

A UserName is a representation of an authorization identifier using the SQL rules for regular and delimited identifiers and thus has the case folding rules etc.

In SQL this behaviour, mapping UserName to authorization identifier is defined by the standard.

In Java code and Java property files Derby chose to use SQL identifier rules (ie. UserName) as the representation of an authorization identifier. Looking back this maybe was a poor choice, a direct representation of the authorization identifier might have been better, (ie. jdbc:derby:cs;user=fred and jdbc:derby:cs;user=FRED would connect as different authorization ids, today they map to the same authorization id). So if we want to be consistent with other Java uses, the policy file should probably use UserName, however using authorization identifier might be clearer.

> 1) UserName -- this is part of the credentials passed to the authentication service. This could be case-sensitive or case-insensitive depending on the rules of the authentication service.

I think the last sentence is incorrect. The rules of UserName are set by Derby, not an arbitrary implementation of the authentication service.
As above, the rules for how UserName map to a unique identity follows the rules of SQL identifiers, and thus UserNames of FRED and fred always map to the same unique identity FRED. However, I'm not sure you agree with this, since you are promoting authorization identifier as a SQL only concept. That may be a valid approach, but I think you need to clearly state the rules for that, and then we can discuss if having two different models (one for database and one for system) adds any benefit or adds complication. The rules would

need to state how UserName maps to a unique identity in a system context.

107. **djd:** Rick Hillegas (JIRA) wrote:

> Those arguments are case sensitive, so it is up to the UserAuthenticator to determine the rules it wants to enforce.

But is such a class honouring the UserAuthenticator api when it does that? The description for authenticateUser states that an unquoted user name is treated as a "case-insensitive authorization identifier" by Derby's authorization system. While this is technically incorrect, (see DERBY-3334), the intention behind the text is to follow SQL identifier rules for converting the parameter userName to a unique authorization identifier within Derby. This can be seen by the fact this mapping will be followed:

userName (in authenticateUser) -- VALUES CURRENT_USER

myHumbleSelf -- MYHUMBLESELF
MYHUMBLESELF -- MYHUMBLESELF
"myHumbleSelf" -- myHumbleSelf

108. **rhillegas:** Thanks for the continued discussion, Dan. I agree that we seem to be talking past one another.

» 2) AuthorizationID -- this is the SQL concept of identity. This is case-insensitive unless double-quoted.
>
>We seem to be using the same terms for slightly different concepts.
>
>I believe I'm using authorization identifier in the way that the SQL standard defines it. This would be a case-sensitive value that defines a unique identity in the sql system (database). Thus an authorization identifier is never double >quoted, contradicting your statement 2).

I think the relevant section of the SQL spec is part 2 section 5.4 (names and identifiers). In that section <authorization identifier> can resolve to be a <user identifier> which in turn can resolve to an <identifier> which in turn can resolve to a <regular identifier> or a <delimited identifier>. The <regular identifier> is uppercased to a normalized value regardless of its original casing and the <delimited identifer> is quoted and is not uppercased. This is what I mean by "This is case-insensitive unless double-quoted".

I think the disconnect here is that the authorization identifier can be referenced in two ways. When it is used in a SQL statement and resolves to a <regular identifier>, it is normalized to uppercase and so, as I would put it, it is case-insensitive. When it is referenced in the form stored in the system catalogs, it is case-sensitive.

>
>A UserName is a representation of an authorization identifier using the SQL rules for regular and delimited identifiers and thus has the case folding rules etc.

The term UserName is a word I invented for this discussion. But it pretty much corresponds to userName as used in the javadoc for UserAuthenticator.

>

>In SQL this behaviour, mapping UserName to authorization identifier is defined by the standard.

This is where I am not tracking you. The term UserName does not occur in the SQL standard as far as I know. It is a term which I thought I invented in order to bring some precision to this discussion. But I don't seem to have succeeded. Can you refer me to the section of the SQL standard which defines this term (or a related term)? That will help me propose some new language which we can agree on.

>
>In Java code and Java property files Derby chose to use SQL identifier rules (ie. UserName) as the representation of an authorization identifier. Looking back this maybe was a poor choice, a direct representation of the authorization i>dentifier might have been better, (ie. jdbc:derby:cs;user=fred and jdbc:derby:cs;user=FRED would connect as different authorization ids, today they map to the same authorization id). So if we want to be consistent with other Java us>es, the policy file should probably use UserName, however using authorization identifier might be clearer.
>
» 1) UserName -- this is part of the credentials passed to the authentication service. This could be case-sensitive or case-insensitive depending on the rules of the authentication service.
>
>I think the last sentence is incorrect. The rules of UserName are set by Derby, not an arbitrary implementation of the authentication service.
>As above, the rules for how UserName map to a unique identity follows the rules of SQL identifiers, and thus UserNames of FRED and fred always map to the same unique identity FRED. However, I'm not sure you agree with this, sinc>e you are promoting authorization identifier as a SQL only concept. That may be a valid approach, but I think you need to clearly state the rules for that, and then we can discuss if having two different models (one for database and o>ne for system) adds any benefit or adds complication. The rules would need to state how UserName maps to a unique identity in a system context.
Ha! Once we can agree on some terms, I can take another crack at stating the rules.

Thanks.

109. **djd:** On the current patch itself I have some questions:

1) Since J2ME/CDC/Foundation does not support some of the security classes used, how is this being handled? Seems like it isn't at the moment. I think a new abstract method is needed in InternalDriver called shutdownCheck() is needed that would do nothing in J2ME but call the checks (through SecurityUtil in J2SE.

2) Any thought on backwards compatibility for the network shutdown command in terms of the DRDA protocol? Seems like the on-wire format for the shutdown command has changed by adding in the user and password fields. What happens if an old version of Derby tries to shutdown a 10.4 server?

3) I'd like to see more comments around the use of doAsPrivileged (e.g. why is this needed rather than doAs, why pass in a null ACC
and the nesting of doAsPrivileged in a doPrivileged call (why is this needed). I'm not sure that the code is using these correctly and given this is security code we need to understand why one is used instead of the other, rather than a comment like "doAs is not strong enough" (in the test).
My gut feeling is that doAsPrivileged is correct for the network server shutdown only, seems

like doAs is needed in the embedded calls if this piece of the functional spec is to be true: Because we use Java Security to model system privileges, the shutdownEngine and create privileges can be granted to code as well as to users.

110. **rhillegas:** » Those arguments are case sensitive, so it is up to the UserAuthenticator to determine the rules it wants to enforce.
>
>But is such a class honouring the UserAuthenticator api when it does that? The description for authenticateUser states that an unquoted user name is treated as a "case-insensitive authorization i>dentifier" by Derby's authorization system. While this is technically incorrect, (see DERBY-3334), the intention behind the text is to follow SQL identifier rules for converting the parameter userName to a >unique authorization identifier within Derby. This can be seen by the fact this mapping will be followed:

The javadoc for UserAuthenticator states the rules for mapping the userName onto the value of SYSSCHEMAS.AUTHORIZATIONID. Indirectly, it warns the customer that Edward and EdWard will be thrown into the same schema even though the company-wide authentication service recognizes these two names as different individuals with different credentials. However, the javadoc should probably punch up the significance of this behavior.

In the Developer's Guide section titled "Example of setting a user-defined class", the sampe code shows a user-supplied authenticator which treats userName as a case-sensitive string. In that example, Edward and EdWard have separate credentials.

At least as far as I can see, the surrounding sections of the Developer's Guide do not explain that Edward and EdWard will be thrown into the same schema. Probably, we should state this explicitly.

111. **djd:** Right I was seeing authorization identifier as a value that resulted from applying the rules of section 5.2/5.4 to the identifiers, but strictly speaking that's not true. Then as Dag points out we are really talking about user identifier for authentication, though possibly authorization identifier for authorization.

So I think we can agree that a user has a unique identity, can we call this UID? This corresponds to what is stored in the system tables.

So a user identifier is a mechanism of representing a UID in a SQL statement using SQL regular or delimited identifiers. Section 5.2/5.4 provides the rules for this mapping.

Derby's Java code and property files use the same syntax rules as SQL's user identifier to represent a UID.

So the concept of UserName and user identifier are actually the same, so we should drop UserName and use user identifier.
[so that's what I meant by the SQL standard defined the rules from user name to authorization identifier - which is really user identifier to UID]

authorization identifier is defined by the SQL spec and can represent a user identifier or a role name.

So if SystemPrincipal represents a user then its input format should be that of a user identifier. I think if we ever wanted to add role support to the policy file then a RolePrincipal would be clearer than having roles and users mixed in the same Principal class.

Rick I think you seem to be saying that a system authentication implementation could treat regular user identifiers of fred and FRED as different. This then contradicts the very definition of user identifier, so it would need a new set of terms to describe this. This seems to go against the existing definitions.

112. **djd:** > The javadoc for UserAuthenticator states the rules for mapping the userName onto the value of SYSSCHEMAS.AUTHORIZATIONID.

Which version (svn revision) are you looking at? I don't see the words SYSSCHEMAS or AUTHORIZATIONID in the javadoc at all.

I do see this text in the section on stating how the userName is handled

"... within the Derby user authorization system"

revision 613345

That's doesn't state it's only for database authorization.

> However, the javadoc should probably punch up the significance of this behavior.I entered DERBY-3334 to make this clearer.

Note the introduction to MyAuthenticationSchemeImpl says "very simple example" :-)
It agrees with the discussion a few pages later
http://db.apache.org/derby/docs/dev/devguide/cdevcsecure24458.html
that says if the external scheme is case-sensitive (as in MyAuthenticationSchemeImpl) then you must always log in with a user name that maps to the value defined in the external scheme.
It remains silent on what to do if the external schema has different identities that map to the same SQL identity, as you say that could be improved.

Later on in that page it says when talking about delimited identifiers:
"(Derby knows to remove the double quotes when passing the name to the external authentication system.)"
I don't believe that's true, at least it contradicts UserAuthenticator's documentation.

113. **djd:** Just to add to the confusion, the derby documentation uses authorization identifier in most cases to mean UID. I think that is what was driving my line of thinking to use authorization identifier when I really meant the UID.

http://db.apache.org/derby/docs/dev/devguide/cdevcsecure37241.html

114. **rhillegas:** >So I think we can agree that a user has a unique identity, can we call this UID? This corresponds to what is stored in the system tables.
I'm afraid that when I read these words, they sound like this to me: "A person has a unigue identity, a UID, which is stored in the system tables." I don't agree with this. Two persons (Edward and EdWard in my example) end up with the same identity. The identity is not unique. That's a problem.

As a practical matter, I don't think that we can please everyone:

1) A customer whose authentication service enforces the case sensitivity of usernames is

probably going to want to grant privileges to case-sensitive names. This customer is not going to be happy if the payroll clerk EdWard gets the shutdown privilege intended for the system administrator Edward.

2) On the other hand, a customer whose authentication service treats Edward and EdWard as the same username is not going to want to have to grant shutdown privilege to every casing combination.

Maybe we could add a userNamesAreCaseSensitive() method to UserAuthenticator or create a CaseSensitiveUserAuthenticator interface to extend UserAuthenticator? Given a case-sensitive UserAuthenticator, we would not have to throw Edward and EdWard into the same schema. The default behavior would be the current behavior. And the default behavior for SystemPrincipal would be, as Dan suggests, that usernames are case-insensitive.

This distinction could be added later on. I don't see that we have to support case-sensitive usernames in 10.4.

115. **djd:** > So I think we can agree that a user has a unique identity, can we call this UID? This corresponds to what is stored in the system tables.

    An alternate way of stating this is:
    an user identifier has a canonical representation which defines uniqueness within Derby's authorization schemes. When storing the user identifier for a user Derby uses the canonical representation.
    [the canonical representation seems to be also called the common normal form or CNF in the SQL standard]

116. **djd:** > Maybe we could add a userNamesAreCaseSensitive() method to UserAuthenticator or create a CaseSensitiveUserAuthenticator interface to extend UserAuthenticator? Given a case-sensitive UserAuthenticator, we would not have to throw Edward and EdWard into the same schema. The default behavior would be the current behavior. And the default behavior for SystemPrincipal would be, as Dan suggests, that usernames are case-insensitive.----------
    ----

    DERBY-3335 proposes a mechanism for an authentication implementation to get the canonical user identifier of the provided user name without changing any api. That's not exactly what you are proposing, I think you are proposing a new way of providing a user name to Derby in Java/JDBC where the user name provided has to match the canonical representation of a user identifier. Obviously in SQL (e.g. GRANT) such user names would have to be delimited.
    Obviously such a change applies to more than authentication, connection level authorization would need to follow the same scheme, the mapping of the provided user name to the SQL user identifier etc.

    I have the feeling that this might confuse the user name situation more that simplify it. Now a client application needs to know how a specific database is treating user names before it can format a connection request. Currently there is a single rule of how user names in Java map to SQL user identifiers.

    If such a feature can be delayed until there is some actual need for it, then I think it just resolves to a simple choice:
    A) name for SystemPrincipal is a user identifier (supports regular and delimited identifiers) and thus matches all other places where a user name is specified in a Java/JDBC context.

    B) name for SystemPrincipal is the canonical representation of an user identifier, which

leads to easier to read policy files but does not match existing Java/JDBC practice.

[edit: be consistent with user identifier]

117. **rhillegas:** >If such a feature can be delayed until there is some actual need for it, then I think it just resolves to a simple choice:
> A) name for SystemPrincipal is a user identifier (supports regular and delimited identifiers) and thus matches all other places where a user name is specified in a Java/JDBC context.
>
> B) name for SystemPrincipal is the canonical representation of an user identifier, which leads to easier to read policy files but does not match existing Java/JDBC practice.
I'm afraid I don't understand what you are saying here. I am suggesting the following. The following policy file declarations result in the same SystemPrincipal:

SystemPrincipal "Edward"

SystemPrincipal "EdWard"

SystemPrincipal "\"EDWARD\""

All of the following SystemPrincipals are different:

SystemPrincipal "Edward"

SystemPrincipal "\"Edward\""

SystemPrincipal "\"EdWard\""

118. **djd:** That's the same as A.

A) name for SystemPrincipal is a user identifier (supports regular and delimited identifiers) and thus matches all other places where a user name is specified in a Java/JDBC context.

119. **mzaun:**
Please, find attached the updated patch DERBY-2109-09, which reflects most changes and comments made since the previous version (DERBY-2109-08):

1) Renamed class DatabasePrincipal to SystemPrincipal and permission "shutdownEngine" to "shutdown".

2) Changed SystemPrincipal for the handling of user names:

2.1) I verified with both Sun's and IBM's JDK (1.5) that the wildcard syntax for policy files, which isn't that well described in the official Java documentation, is supported:
a) grant principal * * { ... }
which is the same as:

b) grant { ... }
which grants permissions to any user represented by any principal class

c) grant principal org.apache.derby.authentication.SystemPrincipal * { ... }
which is with both JDKs the same as:

d) grant principal org.apache.derby.authentication.SystemPrincipal "WILDCARD_PRINCIPAL_NAME" { ... }
which grants permissions to any declared SystemPrincipal user

2.2) removed support for our special characters *, @, and \, as was suggested.

2.3) changed the server.policy, template.policy etc. for the new wildcard syntax SystemPrincipal * { ... }

2.4) changed code and javadoc to reflect that the rules for Authorization Ids, as described in the UserIdentifiers document, apply to SystemPrincipal names.

I think this feature is only partially complete:

a) the policy
grant SystemPrincipal "EVE" { ... }
now matches user names EVE, eVe, or eve -- as it should.

b) as of now, the policy
grant SystemPrincipal "eve" { ... }
only matches user name eve, not EVE or eVe, so, admins have to declare the policy grants using NORMALIZED names if they want case-insensitive matching of names -- I understand we don't want this restriction. This issue requires somewhat more investigation, the Java Security Runtime seems to be doing unexpected things.

c) the policy grant SystemPrincipal "\"eve\"" { ... }
only matches user name eve, not EVE or eVe -- as it should.

3) removed misleading comments in SystemPrincipal, SystemPermission, and DatabasePermission about the non-localization of RuntimeException messages.

4) removed a backward compatibility issue when running with Java Security but without Authentication. Under this configuration, users now don't have to change their customized policy files, as it was intended from the beginning.

There is an implementation question, though: how to find out that we're running effectively without Authentication?
Some code in EmbedConnection tests for instanceof NoneAuthenticationServiceImpl:
// If authentication is not on, we have to raise a warning if sqlAuthorization is ON
// Since NoneAuthenticationService is the default for Derby, it should be ok to refer
// to its implementation here, since it will always be present.
if (authenticationService instanceof NoneAuthenticationServiceImpl)
usingNoneAuth = true;

However, I thought this somewhat fragile and decided to test in SecurityUtil for the property "derby.connection.requireAuthentication", which I found to be a neccessary condition:
// for backward compatibility skip check for create-db authorization
// if we run without Authentication but with a SecurityManager;
// otherwise, users would have to extend any customized policies
final String reqAuthKey = Property.REQUIRE_AUTHENTICATION_PARAMETER;
final String reqAuthValue = PropertyUtil.getSystemProperty(reqAuthKey);
if (!Boolean.valueOf(reqAuthValue).booleanValue()) {
return;

}

Please, let me know your thoughts.

5) addressed a backward compatibility issue with the NetworkServerControl protocol where the shutdown command had been changed to transmit the user credentials. As Dan had pointed out, older NSC clients would raise exceptions when trying to shut down a newer server. I've incremented the protocol's version number for a clean, early failure with an indicative error message.

I don't think requiring current NSC clients for a new server is a major restriction since the socket connections are local to the machine anyway.

6) I'm still looking into another comment by Dan on the Authorization checks in SecurityUtil, which employ a fresh (=null) AccessControlContext with Subject.doAsPrivileged() instead of storing/maintaining a ACC or using the current thread's AccessControlContext (as with Subject.doAs()).

Thanks, Martin

120. **mzaun:**
Rick, Dan,

for the latest patch, I wanted toupdate and recap the backward compatibility issues we'd discussed earlier on derby-dev:

1) Customers running no Authentication and no Java Security: No compatibility issues.

2) Customers running just with Java Security but no Authentication: No compatibility issues.
The issue Dan pointed out should have been addressed by the latest patch.

3) Customers running just with Authentication but no Java Security:
a) Must provide valid credentials when using NetworkServerControl to shutdown the server.

4) Customers running with BOTH Authentication and Java Security:
a) Must provide valid credentials when using NetworkServerControl to shutdown the server.
b) Must add additional privileges to the Java Security policy file unless the default policy file is used.

4.1) Customers with BOTH Authentication and Java Security AND a user by the name "WILDCARD_PRINCIPAL_NAME" cannot grant System Privileges specific to this user! (since this is the reserved wildcard name used by the Java Security Runtime)

5) Customers with <10.4 NetworkServerControl clients cannot shutdown a 10.4 server (due to a protocol change).

Thanks,
Martin

121. **djd:** > I understand we don't want this restriction. This issue requires somewhat more investigation, the Java Security Runtime seems to be doing unexpected things.
I wonder if this is due to SystemPrincipal.getName() not returning the defined string format of SystemPrincipal?

122. **djd:** Some comments on the patch, thanks for working on it and addressing the issues:

- SecurityUtil should be moved out of the o.a.d.security package since it is not part of the external api, how about o.a.d.iapi.security?

- Since the format of the directory path in the create database permission matches FilePermission's format, why not use FilePermission to evaluate it rather than repeating the logic?

>2) Customers running just with Java Security but no Authentication: No compatibility issues.
> The issue Dan pointed out should have been addressed by the latest patch.
I pointed out that it didn't match the spec, but I believe that the previous behaviour was correct. I don't think any other java security check is not enforced if there is a security manager, and I think it's required in order to support being able to grant the shutdown permission to code and Principals that are not SystemPrinicpals as described in the spec.

123. **rhillegas:** Hi Martin,

I have applied the 09 patch and run the regression tests. I am seeing 7 instances of the following error:

1) testSystemShutdown(org.apache.derbyTesting.functionTests.tests.jdbcapi.AuthenticationTest)junit.framework.C
Unexpected SQL state. expected:<[XJ015]> but was:<[08004]>

124. **mzaun:**
Just a quick update:
a) I'm able to reproduce the AuthenticationTest failures and are working on it.
b) Per derby-dev discussion "JMX meeting system authorization (DERBY-2109 & 1387)", I've made the System Privileges checks only dependent upon the presence of a SecurityManager, not upon Authentication.
c) I'm debugging a described issue with the normalization of names (non-normalized names in policy files not properly matched by authorization checks).
Hope to have these issues resolved soon and will propose another patch update then.
Martin

125. **djd:** Is a new version of the patch likely soon? Some of the SecuirtyUtil code is useful for JMX. If not, I'll extract the useful code and create SecurityUtil in o.a.d.iapi.security.

126. **djd:** With the additional system permissions proposed in DERBY-3462 I wonder if it makes sense to change the style of names & actions in SystemPermission.

Today a "shutdown" name is proposed and potential for future "shutdownEngine" and "shutdownServer" with no actions.

DERBY-3462 is proposing names of jmxControl, serverControl, engineControl etc also with no actions.

Looking at the standard Permission class it seems the name is meant to represent an object that the permission applies to and action represent actions on that object.
Thus it would seem to make more sense and be consistent with other Permissions to have:

name=server action=control | monitor | shutdown
name=engine action=control | monitor | shutdown

name=jmx action=control

Not sure what the current "shutdown" would map to (note it is an action, but defined as a name), it could be:

name="system" action="shutdown" => implies engine,shutdown & server,shutdown

This could be changed after any updated patch is applied, but would need to be done before any release.

127. **mzaun:** Please find attached patch update DERBY-2109-10 (replacing DERBY-2109-09):

a) Fixed AuthenticationTest (and other) failures; junit-all and derbyall run fine with the patch on my machine.

b) Made the System Privileges checks only dependent upon the presence of a SecurityManager, not upon Authentication, per derby-dev discussion "JMX meeting system authorization (DERBY-2109 & 1387)".

c) Moved class SecurityUtil from o.a.d.security to o.a.d.iapi.security, per suggestion by Dan.

d) Investigated an issue with non-normalized names in policy files where a grant to user edWard would not give permissions to the authenticated users edward or EDWARD but edWard only.

Bottomline: When evaluating permissions, the (Sun) Java Security Runtime uses the principal names as found literally in the policy file and not as returned by SystemPrincipal.getName() (where we could return normalized names).

As a workaround, our class SystemPrincipal could implement the non-standard interface com.sun.security.auth.PrincipalComparator, which declares a method implies(Subject) that allows for Principals to match Subjects based on normalized Authorization Identifiers, e.g., a policy grant to edWard would give permissions to all users edWard, edward, and EDWARD. But then we'd be relying upon a non-standard Security Runtime class and behaviour. (Yet another approach would be to construct the Subject with a Principal list having all lower/upper case combinations of an identifier, but that's clearly not feasible.)

I changed classes SecurityUtil and SystemPrincipal and added javadoc comments reflecting that
- SystemPrincipal names are not Authorization Identifiers (i.e., non-delimited identifiers handled case-insensitively by magic of SystemPrincipal);
- SecurityUtil constructs a Subject with two SystemPrincipals: one with the literal user name and another with the normalized name.

For users who want to put specific principal grants into customized policy files this means:
- a grant to a principal named EDWARD gives permissions to all users edward, edWard, and EDWARD (and all other lower/upper case spellings);
- a grant to principal edWard only gives permissions to the user edWard.

In my view that's still an acceptable and documentable behaviour; but as said, I haven't found any other feasible workaround lifting the Java Security Runtime's literal handling of principal names.

e) Added comments to SystemPrivilegesPermissionTest.java on the use of Subject.doAsPrivileged()

versus doAs() after further experiments with the Java Security Runtime. Bottomline: doAsPrivileged() is the only implementation option for our purposes.

f) The patch does not reflect the latest suggestions by Dan for the shutdown permission following a new naming scheme (wanted to get this patch out and tested first). But will look into it and comment.

Martin

128. **djd:** > Bottomline: When evaluating permissions, the (Sun) Java Security Runtime uses the principal names as found literally in the policy file and not as returned by SystemPrincipal.getName() (where we could return normalized names).
It's hard to see how that is the case, since the policy file is read in and converted to a Policy object containing permissions and for Derby's SystemPermission the class must be o.a.d.security.SystemPermission.

    Could you confirm the format for names in the policy file in the patch? Does it match the description here:
    https://issues.apache.org/jira/browse/DERBY-2109?focusedCommentId=12561537#action_12561537

    - does it support delimited identifiers of the form "\"fred@acme.com\""?
    - is the only issue that non-delimited identifiers "fred" do not resolve correctly?

129. **djd:** I'm prepared to commit this patch but I think there are a number of follow changes that would need to be done before a 10.4 release is made. I guess if it's committed to trunk then it can always be backed out from the branch if it doesn't seem ready for release.

    - figuring out the principal names in the policy files - I'd be interested to see the implementation of SystemPrincipal you used to try and implement the required functionality

    - related to the previous one is having more time to understand this:
    + // An alternative approach of normalizing all names within
    + // SystemPrincipal has issues; see comments there.
    + principals.add(new SystemPrincipal(user));
    + principals.add(new SystemPrincipal(getAuthorizationId(user)));

    From a security point of view does this have the potential to allow one user to piggy back on the permissions of another user?

    - ensuring the new security objects that are serializable have serialization ids to ensure compatibility across releases

    - making the new security objects final

    Thanks for working on the functionality through many patches!

130. **djd:** Not sure the patch can be committed. The code uses javax.security classes which are not available on J2ME/CDC/Foundation.

    I raised this earlier:

    1) Since J2ME/CDC/Foundation does not support some of the security classes used, how is this being handled? Seems like it isn't at the moment. I think a new abstract method is needed in InternalDriver called shutdownCheck() is needed that would do nothing in J2ME

but call the checks (through SecurityUtil in J2SE.
https://issues.apache.org/jira/browse/DERBY-2109?focusedCommentId=12561087#action_12561087

Does the patch work currently on J2ME, I don't have any environment to check it out at the moment. If someone has a J2ME environment and try the patch and see if shutting down Derby works that would be a help.

131. **mzaun:**
I'm sorry to say that I overlooked one line of debug code in SecurityUtil that made it into DERBY-2109-10. (I'd forgotten tagged that line as usual.) Unfortunately, the line is significant since the published code checks for a "dummy" permission I'd put in there for doAs() v. doAsPrivileged() debugging purposes:
+ //AccessController.checkPermission(perm);
+ AccessController.checkPermission(new java.util.PropertyPermission("user.dir", "read"));

I'm right now running junit-all (1 error) and will publish another patch update right after.

> Daniel John Debrunner wrote:
> > Bottomline: When evaluating permissions, the (Sun) Java Security Runtime uses the principal names as found literally in the policy file and not as returned by SystemPrincipal.getName() (where we could return normalized names).
> It's hard to see how that is the case, since the policy file is read in and converted to a Policy object containing permissions and for Derby's SystemPermission the class must be o.a.d.security.SystemPermission.

It's correct that we're using our SystemPermission classes only -- but the issue is in the realm of Subject/Principal checking by the Java Security Runtime when matching the Subject instantiated by our code against the Principal declaration in the policy file. This check is not entirely carried out by means of getName()/equals()/hashCode() on our SystemPrincipal class, in which case we could have had the name comparisons always done on the normalized names. Instead, the Java Security Runtime uses the literal Principal name as declared in the policy file and denies permission when it can't find an exact match in our Subject's SystemPrincipal list.

> - figuring out the principal names in the policy files - I'd be interested to see the implementation of SystemPrincipal you used to try and implement the required functionality
The SystemPrincipal class in DERBY-2109-09 attempted to encapsulate Authorization Identifiers and "normalized" the principal name right within the constructor, so that getName()/equals()/hashCode() would operate on (normalized) auth ids.
As commented I also looked into SystemPrincipal implementing com.sun.security.auth.PrincipalComparator by adding an implies(Subject) method returning true when a the Subject's principal list contains a normalized name matching this principal's normalized name. With a few more tricks I got it working -- but that was just for insight, since PrincipalComparator is a non-standard interface.

> - related to the previous one is having more time to understand this:
> + // An alternative approach of normalizing all names within
> + // SystemPrincipal has issues; see comments there.
> + principals.add(new SystemPrincipal(user));
> + principals.add(new SystemPrincipal(getAuthorizationId(user)));
Adding the normalized identifier ensures that a non-delimited principal "FRED" matches users "fred", "Fred", "FRED" ...

Let me know if you think the code needs more comments.

> Could you confirm the format for names in the policy file in the patch? Does it match the description here:
https://issues.apache.org/jira/browse/DERBY-2109?focusedCommentId=12561537#action_12561537
> - does it support delimited identifiers of the form "\"fred@acme.com\""?

yes

> - is the only issue that non-delimited identifiers "fred" do not resolve correctly?

They resolve incompletely: non-delimited identifies "fred" only matches "fred", while "FRED" matches "fred", "Fred", "FRED" ...

Here's what I've verified. A policy file with belows grants results in the following System Privileges behaviour for users:

grant principal org.apache.derby.authentication.SystemPrincipal "MARTIN" { ... }
checks for users:
martin -- granted
marTin -- granted
MARTIN -- granted
"marTin" -- denied, missing permission (delimited identifier, different from marTin)

grant principal org.apache.derby.authentication.SystemPrincipal "edWard" { ... }
checks for users:
edWard -- granted
edward -- denied, missing permission

grant principal org.apache.derby.authentication.SystemPrincipal "\"fred@acme.com\"" { ...
}
checks for users:
"fred@acme.com" -- granted
"Fred@ACME.COM" -- denied, missing permission
fred@acme.com -- denied, illegally formed name as complained by IdUtil.getUserAuthorizationId() because it's a non-delimited identifier having special characters

However, in the last case the exception is not nicely presented to the client and needs improvement (I'd expected this case not to pass authentication, but we have to cover for it, especially since we now can have authorization checks without prior authentication).

> From a security point of view does this have the potential to allow one user to piggy back on the permissions of another user?

Not really. Under the rules for Identifiers for authentication and authorization, the non-delimited identifiers "fred", "Fred", "FRED" all represent the same person. So, if user "fred" can piggy-back on a principal grant to "FRED" -- that's the requested feature that took some effort to implement :)

In contrast, user "marTin" cannot piggy-back on a principal grant for "\"martin\"".

> - ensuring the new security objects that are serializable have serialization ids to ensure compatibility across releases

I'm not aware anymore why SystemPrincipal implements Serializable, so, perhaps, I should research and add a comment. I'm not sure there's a good reason for this class to be serializable (to the contrary, often information about user names etc should not be serialized), but if there is, I agree, a serialization id should be there (with the class being so simple, we probably won't need readObject() and writeObject()).

Our Permission classes are not serializable, so, I don't think we have other security objects to think about.

While NetworkServerControlImpl also does not implement Serializable, I wonder if the userArg, passewordArg, and bootpasswordArg fields should be declared transient as good pactice/precaution.

> - making the new security objects final

Yes, will do for SystemPrincipal, DatabasePermission and SystemPermission.

Martin

132. **rhillegas:** I'm waiting for the next rev of the patch before I start running unit tests. If the tests pass, I'm inclined to check this in in order to get the Replication and JMX folks unstuck. I'm hoping that we can treat the J2ME issue like the other issues: address it in a follow-on patch. Perhaps Martin could boost the priority of the J2ME issue. If we have to back this work out, on the branch, that's fine. However, I hope that our release manager will be willing to delay the release a bit if it looks like a little more effort could fix these outstanding issues.

133. **djd:** The trouble is that the patch will break J2ME, all the other issues are related to this specific new code and do not have such a huge effect.

    Would we commit a patch that broke Java SE 6?

134. **rhillegas:** I would allow a patch which didn't work on Java 6 if it got other people unstuck and if I trusted the developer to submit a fix soon.

135. **mzaun:**
    > Daniel John Debrunner wrote:
    > 1) Since J2ME/CDC/Foundation does not support some of the security classes used, how is this being handled? Seems like it isn't at the moment. I think a new abstract method is needed in InternalDriver called shutdownCheck() is needed that would do nothing in J2ME but call the checks (through SecurityUtil in J2SE.

    The System Privileges checks are invoked from three places: NetworkServerControlImpl, InternalDriver, and EmbedConnection (and there may be more in future). They all call into SecurityUtil. Instead of introducing abstract methods in three different places, we may want to think about using SecurityUtil as a central switch for checking/not checking SystemPrivileges.

    Right now SecurityUtil is is a static utility class, which doesn't support delegation. I see the following options:
    a) Make SecurityUtil an interface and (perhaps rename it) and have a public singleton class providing an instance appropriate to the environment (i.e., dummy implementation for J2ME).
    b) Keep SecurityUtil as class but make its method non-static, abstract and introduce a static method getSecurityChecker() (or so) returning a singleton instance implementing these methods according to the environment.

c) Keep SecurityUtil as utility class but have the static methods internally delegate to an instance appropriate to the environment.

I like c) best since it keeps the callers unchanged. Any thoughts/comments welcome.

If you have a pointer to a derby code example for me where switch implementation classes to accommodate J2ME, that would be helpful to me too.

Thanks,
Martin

136. **mzaun:** Please find attached for review and testing the patch update DERBY-2109-11 (replacing DERBY-2109-10):

a) Fixed the debug/dummy code oversight in SecurityUtil, now really doing: AccessController.checkPermission(perm).

b) All unit tests now passing (fixed a few unit tests and DRDAServerStarter).

c) Made public API security classes final (SystemPrincipal, DatabasePermission, SystemPermission) per suggestion by Dan.

d) Added serialization support (serialVersionUID) and comments to SystemPrincipal.

Starting a final derbyall and junit-all run overnight,
Martin

137. **mzaun:** > Starting a final derbyall and junit-all run overnight,
No errors.
Martin

138. **djd:** > The System Privileges checks are invoked from three places: NetworkServerControlImpl, InternalDriver, and EmbedConnection

The network server doesn't currently run under J2ME and the other two places are already factored into platform specific modules (J2ME, JDK1.4/1.5, Java SE 6) so a quick fix would be to take advantage of the existing factoring and just have security check methods in InternalDriver that either call SecurityUtil or don't for J2ME.

139. **mzaun:**
> The network server doesn't currently run under J2ME and the other two places are already factored into platform specific modules (J2ME, JDK1.4/1.5, Java SE 6) so a quick fix would be to take advantage of the existing factoring and just have security check methods in InternalDriver that either call SecurityUtil or don't for J2ME.

Thanks! Here's my understanding of your quick fix proposal:
1) change method InternalDriver.checkShutdownPrivileges() to be abstract
2) move the implementation to Driver20.checkShutdownPrivileges() (this way Driver30, 40 will have it too)
3) define an empty method Driver169.checkShutdownPrivileges()
I'll do that and run tests.

Martin

140. **djd:** Right, and something similar for create database.

    One could even just have an abstract checkPermission(Permission) method in InternalDriver and then pass in the required permission.
    This tends to be more in line with how permission checks are implemented (see the permission in-line rather than hidden in a utility class).
    Then only a single method is added to InternalDriver rather than one per type of permission check.
    This also means SecurityUtil becomes a real utility class rather than having specific knowledge of permission requirements from other modules .

141. **myrna:** When a new patch with a fix for J2ME is available, I'll be happy to run again, but my results with patch 10 are that for derbyall, only 5 tests passed, for suites.All, 113 fixtures out of 2697 failed (didn't run suites.All with jars, only classes).
    For reference, here's a stack example, from lang/locktable.sql's derby.log, showing a problem would've been through ij:

    java.lang.NoClassDefFoundError: javax.security.auth.Subject
    at org.apache.derby.iapi.security.SecurityUtil.createSystemPrincipalSubject(SecurityUtil.java:112)
    at org.apache.derby.iapi.security.SecurityUtil.checkDatabaseCreatePermission(SecurityUtil.java:255)
    at org.apache.derby.impl.jdbc.EmbedConnection.createDatabase(EmbedConnection.java:2259)
    at org.apache.derby.impl.jdbc.EmbedConnection.<init>(EmbedConnection.java:351)
    at org.apache.derby.jdbc.Driver169.getNewEmbedConnection(Driver169.java:57)
    at org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:240)
    at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:406)
    at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:373)
    at java.lang.reflect.AccessibleObject.invokeL(AccessibleObject.java:213)
    at java.lang.reflect.Method.invoke(Method.java:272)
    at org.apache.derby.impl.tools.ij.util.getDataSourceConnection(util.java:426)
    at org.apache.derby.impl.tools.ij.util.startJBMS(util.java:516)
    at org.apache.derby.impl.tools.ij.util.startJBMS(util.java:585)
    at org.apache.derby.impl.tools.ij.ConnectionEnv.init(ConnectionEnv.java:64)
    at org.apache.derby.impl.tools.ij.utilMain.initFromEnvironment(utilMain.java:165)
    at org.apache.derby.impl.tools.ij.Main.<init>(Main.java:230)
    at org.apache.derby.impl.tools.ij.Main.getMain(Main.java:193)
    at org.apache.derby.impl.tools.ij.Main.mainCore(Main.java:178)
    at org.apache.derby.impl.tools.ij.Main.main(Main.java:73)
    at org.apache.derby.tools.ij.main(ij.java:59)

142. **mzaun:**
    Please find attached for review and testing the updated patch DERBY-2109-12, which addresses the J2ME/CDC failures with former patches. Patch #12 passes junit-all on my machine (and I'll run derbyall later tonight).

    Myrna, thanks for testing on J2ME/CDC. It would be great if you could run the latest patch #12 too.

    I've applied the code refactorizations along the lines Dan's suggested:

    a) InternalDriver now has a new method:
    abstract public void checkSystemPrivileges(String user, Permission perm) throws Exception;
    which is implemented in Driver20 with a call to SecurityUtil, and in Driver169 as an empty method body (with a comment).

    b) SecurityUtil has been stripped off knowledge of SystemPermission, DatabasePermission

(but continues to deal with Subjects, Principals, Authorization Ids, and executing the permission checks).

c) InternalDriver, EmbedConnection have methods
public void checkShutdownPrivileges(String user) throws SQLException
public void checkDatabaseCreatePrivileges(String user, String dbname) throws SQLException
both of them calling the driver's (abstract) checkSystemPrivileges().

d) NetworkServerControlImpl's method
public void checkShutdownPrivileges() throws SQLException
calls static SecurityUtil.checkUserHasPermission(). There may be an option to change that call to InternalDriver's checkSystemPrivileges(), but so far (by design, I guess), NetworkServerControl only knows of Driver, not InternalDriver.

Martin

143. **myrna:** I ran derbyall, for starters, with my JSR169 implementation (the jvm available through IBM's weme6.1) and unfortunately, this time, only 4 tests passed...(store/logDevice.sql; store/rollForwardBackup.sql, store/backupRestore.sql, store/bootlock.java). 153 tests failed.

Many test diffs show something like this:
0a1
> Parsing policy file: file:C:/derbyt/svn/tst/j9_f11/d2109_12/derbyall/derby_tests.policy, found unexpected: permission
and no connections are made (no database is created).

Here's one derby.log file example (again, lockTable) shows that this time it's the String.split message which is not supported with JSR169 - at least, not with the implementation I'm using:

java.lang.NoSuchMethodError: java/lang/String.split(Ljava/lang/String;)[Ljava/lang/String;
at org.apache.derby.security.DatabasePermission.initActions(DatabasePermission.java:237)
at org.apache.derby.security.DatabasePermission.<init>(DatabasePermission.java:210)
at org.apache.derby.impl.jdbc.EmbedConnection.checkDatabaseCreatePrivileges(EmbedConnection.java:2326)
at org.apache.derby.impl.jdbc.EmbedConnection.createDatabase(EmbedConnection.java:2261)
at org.apache.derby.impl.jdbc.EmbedConnection.<init>(EmbedConnection.java:354)
at org.apache.derby.jdbc.Driver169.getNewEmbedConnection(Driver169.java:60)
at org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:237)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:406)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:373)
at java.lang.reflect.AccessibleObject.invokeL(AccessibleObject.java:213)
at java.lang.reflect.Method.invoke(Method.java:272)
at org.apache.derby.impl.tools.ij.util.getDataSourceConnection(util.java:426)
at org.apache.derby.impl.tools.ij.util.startJBMS(util.java:516)
at org.apache.derby.impl.tools.ij.util.startJBMS(util.java:585)
at org.apache.derby.impl.tools.ij.ConnectionEnv.init(ConnectionEnv.java:64)
at org.apache.derby.impl.tools.ij.utilMain.initFromEnvironment(utilMain.java:165)
at org.apache.derby.impl.tools.ij.Main.<init>(Main.java:230)
at org.apache.derby.impl.tools.ij.Main.getMain(Main.java:193)
at org.apache.derby.impl.tools.ij.Main.mainCore(Main.java:178)
at org.apache.derby.impl.tools.ij.Main.main(Main.java:73)
at org.apache.derby.tools.ij.main(ij.java:59)

I'll kick off suites.All, to see what that shows us...

144. **myrna:** suites.All (not using any ant target) with my JSR169 implementation with patch 102 installed (and merged with latest changes from trunk) results in 102 failures out of 2697...

101 of those are because of String.split, for example, here is the stack trace from lang.TimeHandlingTest:

---

org.apache.derbyTesting.functionTests.tests.lang.TimeHandlingTestjava.sql.SQLException: Java exception: 'java/lang/String.split(Ljava/lang/String;)[Ljava/lang/String;: java.lang.NoSuchMethodError'.
at org.apache.derby.impl.jdbc.SQLExceptionFactory.getSQLException(SQLExceptionFactory.java:45)
at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Util.java:88)
at org.apache.derby.impl.jdbc.Util.javaException(Util.java:245)
at org.apache.derby.impl.jdbc.TransactionResourceImpl.wrapInSQLException(TransactionResourceImpl.java:403)
at org.apache.derby.impl.jdbc.TransactionResourceImpl.handleException(TransactionResourceImpl.java:346)
at org.apache.derby.impl.jdbc.EmbedConnection.handleException(EmbedConnection.java:2023)
at org.apache.derby.impl.jdbc.EmbedConnection.<init>(EmbedConnection.java:511)
at org.apache.derby.jdbc.Driver169.getNewEmbedConnection(Driver169.java:60)
at org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:237)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:406)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(EmbeddedSimpleDataSource.java:373)
at org.apache.derbyTesting.junit.DataSourceConnector.openConnection(DataSourceConnector.java:67)
at org.apache.derbyTesting.junit.TestConfiguration.openDefaultConnection(TestConfiguration.java:1312)
at org.apache.derbyTesting.junit.BaseJDBCTestSetup.getConnection(BaseJDBCTestSetup.java:72)
at org.apache.derbyTesting.junit.CleanDatabaseTestSetup.setUp(CleanDatabaseTestSetup.java:104)
at junit.extensions.TestSetup$1.protect(TestSetup.java:18)
at junit.extensions.TestSetup.run(TestSetup.java:23)
at org.apache.derbyTesting.junit.BaseTestSetup.run(BaseTestSetup.java:57)
Caused by: java.lang.NoSuchMethodError: java/lang/String.split(Ljava/lang/String;)[Ljava/lang/String;
at org.apache.derby.security.DatabasePermission.initActions(DatabasePermission.java:237)
at org.apache.derby.security.DatabasePermission.<init>(DatabasePermission.java:210)
at org.apache.derby.impl.jdbc.EmbedConnection.checkDatabaseCreatePrivileges(EmbedConnection.java:2326)
at org.apache.derby.impl.jdbc.EmbedConnection.createDatabase(EmbedConnection.java:2261)
at org.apache.derby.impl.jdbc.EmbedConnection.<init>(EmbedConnection.java:354)
... 21 more

The last failure is different - I think it ok if this one test gets excluded from the unitTests suite for JSR169.

---

102) testSystemPrivileges(org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest)java.lang.NoCla
javax.security.auth.Subject
at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest$RunAsPrivilegedUserAction.run(Syste
at java.security.AccessController.doPrivileged(AccessController.java:191)
at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.execute(SystemPrivilegesPermissionTes
at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.checkSystemPermission(SystemPrivileg
at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.execute(SystemPrivilegesPermissionTes
at org.apache.derbyTesting.unitTests.junit.SystemPrivilegesPermissionTest.testSystemPrivileges(SystemPrivilegesP
at java.lang.reflect.AccessibleObject.invokeV(AccessibleObject.java:205)

at org.apache.derbyTesting.junit.BaseTestCase.runBare(BaseTestCase.java:101)
at junit.extensions.TestDecorator.basicRun(TestDecorator.java:22)
at junit.extensions.TestSetup$1.protect(TestSetup.java:19)
at junit.extensions.TestSetup.run(TestSetup.java:23)
---

145. **djd:** For patch 12 could you point to where these new features are tested as functional tests:

    - passing authentication for network server shutdown
    - system level authorization for system shutdown
    - system level authorization for database creation

    I stared at the patch for a while but couldn't see any new tests (apart from the unit tests which I think are only testing the behaviour of the low-level permissions and principals.

146. **rhillegas:** Thanks for the patch, Martin. For the record, the unit tests ran fine for me on Java 5 on my Mac, except for the known regressions mentioned in http://www.nabble.com/failures-in-org.apache.derbyTesting.functionTests.tests.lang.LangScripts-to15747036.html#a15747036 (a change in the text for error 42818) and the known regression in TransactionTable. However, given the J2ME issues, the patch does not seem ready for commit.

147. **myrna:** I woke up last night realizing I shouldn't have merged first.
    I verified this morning, and this patch does *not* use String.split.

    Instead, something else has gone in that caused these failures. They'll show up in the nightlies for this platform...
    I'm sorry for the confusion. I will run with just the patch, but am woefully out of time today. Hopefully someone else can track down what went in between the revision of the patch and 632207.

148. **myrna:** The plot thickens.
    Sync-ed up to revision 631208, without the patch, with my J2ME things pass (similarly to as reported inhttp://people.apache.org/~myrnavl/derby_test_results/main/windows/testSummary-631820.html)

    However, sync-ed up to this (631208, the level of the patch) with the patch
    I get the erros I reported above.

    Sync-ed up to 632212 without the patch also fails...but differently, so that's another issue.

    The trouble indicated above holds - I'm sorry to say.

149. **kmarsden:** The J2ME run had other problems, perhaps related to:
    r632125 | djd | 2008-02-28 13:43:25 -0800 (Thu, 28 Feb 2008) | 5 lines

    DERBY-3445 Adds ant targets to run the junit-all tests with EMMA code coverage.
    Fixes some permission issues in tests when running coverage with EMMA.
    DERBY-3153 Allows the junit-all tests to be run with ant 1.7
    .....Parsing policy file: jar:file:/C:/jartest/classes/derbyTesting.jar!/org/a
    pache/derbyTesting/functionTests/util/derby_tests.policy, found unexpected: perm
    ission
    EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE.EEE.EEEE.EE.EEEEEEEE.I
    E.E.E.E.E.E.EEE.E.E.E.EEEEEE.E.E.E.E.E.E.E.E.EEEEEEEEEEEEEEEEEEEEEEEEEEEE.EEEEEE.
    EEE.E.E.E.E.E.

E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.E.
E.
E.E.E.E.E.E.E.E.E.E.E.E.E.E.EEEE.E.EEEEEEEEEEEEEEEEEEEjava.sql.SQLException: Java exception: 'Access denied (java.util.PropertyPermission user.dir read): java.sec urity.AccessControlException'.
at org.apache.derby.impl.jdbc.SQLExceptionFactory.getSQLException(Unknow n Source)
at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Unknown Source)
at org.apache.derby.impl.jdbc.Util.javaException(Unknown Source)
at org.apache.derby.impl.jdbc.TransactionResourceImpl.wrapInSQLException (Unknown Source)
at org.apache.derby.impl.jdbc.TransactionResourceImpl.handleException(Un known Source)
at org.apache.derby.impl.jdbc.EmbedConnection.handleException(Unknown So urce)
at org.apache.derby.impl.jdbc.EmbedConnection.<init>(Unknown Source)
at org.apache.derby.jdbc.Driver169.getNewEmbedConnection(Unknown Source)
at org.apache.derby.jdbc.InternalDriver.connect(Unknown Source)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(Unknown Source)
at org.apache.derby.jdbc.EmbeddedSimpleDataSource.getConnection(Unknown Source)
at org.apache.derbyTesting.junit.DataSourceConnector.openConnection(Data SourceConnector.java:54)
at org.apache.derbyTesting.junit.TestConfiguration.openDefaultConnection (TestConfiguration.java:1312)
at org.apache.derbyTesting.junit.BaseJDBCTestSetup.getConnection(BaseJDB CTestSetup.java:72)
at org.apache.derbyTesting.functionTests.tests.jdbcapi.SURDataModelSetup .setUp(SURDataModelSetup.java:119)
at junit.extensions.TestSetup$1.protect(TestSetup.java:18)
at junit.framework.TestResult.runProtected(TestResult.java:124)
at junit.extensions.TestSetup.run(TestSetup.java:23)
....

I will take a closer look and file a Jira entry.

As for the split code it was added June 6. revision 544870 as part of DERBY-2109. Is it possible that the new patch causes that code to be exercised for the first time?

150. **mzaun:** Myrna, Rick, thank you very much for testing.

On the NoSuchMethodError: (my) class DatabasePermission DOES call method String.split(String) at line 237. However, the javadoc says this method has been introduced with JDK 1.4. It's not in the patch but in the class that's already been in the trunk for some time (but not invoked yet).

So, I'm confused: does it mean that for J2ME/CDC we can't fully rely upon the 1.4 java.lang API ?

Martin

151. **mzaun:** > As for the split code it was added June 6. revision 544870 as part of DERBY-2109.
> Is it possible that the new patch causes that code to be exercised for the first time?

Yes, that's exactly what happened. Thanks,
Martin

152. **kristwaa:** Martin wrote:

So, I'm confused: does it mean that for J2ME/CDC we can't fully rely upon the 1.4 java.lang API ?

Yes, that is the case. It is based on the Java SE 1.4 API, but it is not the same.
To be sure, I think you have to consult the J2ME API specification, which you can download from JCP I think.

153. **djd:** FYI, the J2ME profile Derby is coded against is:

J2MC/CDC/Foundation 1.1 with JSR 169 for the JDBC subset.

154. **djd:** I'm working on committing portions of the patch to move this issue forward and reduce the amount of effort for everyone in dealing with a new huge patch each time.

First will be the security objects (e.g permissions, principals and SecurityUtil) all of which I think can be committed but continue not to be used by the current code, thus not causing problems. I'm will run junit-all before committing.

The next step may be the changes that relate to adding user & password to network server control.

If I could get an answer on the test changes then I could proceed quicker.

155. **djd:** Committed (revision 632419) a sub-set of patch 12 related to the security code without enabling any functionality.
Note that the SystemPrivilegesPermissionTest is not added into the _Suite (as the complete patch does)
as I think it will fail under J2ME. It passes with classes but fails with jars but that is to be expected
as the Derby's permissions classes are not yet added into the jars, this will occur once they are used.

M java/build/org/apache/derbyBuild/classlister.java
D java/engine/org/apache/derby/authentication/DatabasePrincipal.java
A java/engine/org/apache/derby/authentication/SystemPrincipal.java
A java/engine/org/apache/derby/iapi/security
A java/engine/org/apache/derby/iapi/security/SecurityUtil.java
M java/engine/org/apache/derby/security/DatabasePermission.java
M java/engine/org/apache/derby/security/SystemPermission.java
M java/testing/org/apache/derbyTesting/unitTests/junit/SystemPrivilegesPermissionTest.policy
M java/testing/org/apache/derbyTesting/unitTests/junit/SystemPrivilegesPermissionTest.java

156. **djd:** Committed revision 632452 another sub-set of the 12 patch.
Framework code in InternalDriver and sub-classes to check permissions.
Code to enforce engine shutdown permission in InternalDriver but commented out. Need to

understand which test code in the patch may be affected.
New error messages.

M java/engine/org/apache/derby/jdbc/InternalDriver.java
M java/engine/org/apache/derby/jdbc/Driver169.java
M java/engine/org/apache/derby/jdbc/Driver20.java
M java/engine/org/apache/derby/loc/messages.xml
M java/shared/org/apache/derby/shared/common/reference/SQLState.java
M java/testing/org/apache/derbyTesting/functionTests/tests/lang/ErrorCodeTest.java

157. **djd:** Committed revision 632502 Major sub-set of patch 12 that requires a user name and password for network server shutdown.
Also includes the code to check a permission when shutting down the network server but the actual call to check the permission is commented out.

M java/engine/org/apache/derby/iapi/jdbc/DRDAServerStarter.java
M java/drda/org/apache/derby/impl/drda/NetworkServerControlImpl.java
M java/drda/org/apache/derby/drda/NetworkServerControl.java
M java/drda/org/apache/derby/loc/drda/messages_en.properties
M java/testing/org/apache/derbyTesting/functionTests/tests/jdbcapi/AutoloadTest.java
M java/testing/org/apache/derbyTesting/functionTests/tests/derbynet/ServerPropertiesTest.java
M java/testing/org/apache/derbyTesting/functionTests/tests/derbynet/NSSecurityMechanismTest.java
M java/testing/org/apache/derbyTesting/functionTests/tests/derbynet/SecureServerTest.java
M java/testing/org/apache/derbyTesting/functionTests/master/maxthreads.out
M java/testing/org/apache/derbyTesting/functionTests/master/derbyrunjartest.out
M java/testing/org/apache/derbyTesting/functionTests/master/timeslice.out
M java/testing/org/apache/derbyTesting/junit/NetworkServerTestSetup.java

158. **army:** A few javadoc warnings that have arisen from the recent commits:

[javadoc] java\engine\org\apache\derby\iapi\jdbc\DRDAServerStarter.java:107: warning - @param argument "address" is not a parameter name.

[javadoc] java\engine\org\apache\derby\iapi\jdbc\DRDAServerStarter.java:107: warning - Tag @see: reference not found: NetworkServerControl

[javadoc] java\engine\org\apache\derby\authentication\SystemPrincipal.java:47: warning - Tag @see: can't find name in java.security.Principal

159. **myrna:** I think one or more of the commits may have caused weme6.1 (my J2ME/JSR169 implementation) to run into 152 failed tests in derbyall.
The last one in the report for example has a failure diff like so:
==========
*** Start: T_MarkedLimitInputStream jdkJ2ME Foundation Specification v1.1 derbyall:unit 2008-03-01 07:05:44 ***
1 del
< -- Unit Test T_MarkedLimitInputStream starting
2 del
< -- Unit Test T_MarkedLimitInputStream finished
2 add
> Parsing policy file: file:C:/jartest/JarResults.2008-02-29/weme6.1_derbyall/derby_tests.policy, found unexpected: permission

> xxxxxxFILTERED-TIMESTAMPxxxxxGMT Thread[main,5,main] java.security.AccessControlException: Access denied (java.io.FilePermission derby.log read)
> xxxxxxFILTERED-TIMESTAMPxxxxxGMT Thread[main,5,main] Cleanup action starting
> ERROR XBM02: Startup failed due to missing functionality for org.apache.derbyTesting.unitTests.harness.UnitT
Please ensure your classpath includes the correct Derby software.
> Cleanup action completed
> xxxxxxFILTERED-TIMESTAMPxxxxxGMT Thread[main,5,main] xxxxxxFILTERED-TIMESTAMPxxxxxGMT Thread[main,5,main] Startup failed due to missing functionality for org.apache.derbyTesting.unitTests.harness.UnitTestManager. Please ensure your classpath includes the correct Derby software.
> ERROR XBM02: Startup failed due to missing functionality for org.apache.derbyTesting.unitTests.harness.UnitT
Please ensure your classpath includes the correct Derby software.
Test Failed.
==========
I understood from the discussion re DERBY-3445 that with this jvm I need to run with properties spelled out (so I'm now running with -Demma.active=""). Is there some property I need to specify for these changes too that I missed?

Any other suggestions?

There is no derby.log, and (so) no stack trace...

160. **djd:** None of the recent commits for this issue (sub-sets of patch 12) changed any policy file, so not sure what's going on with the weme 6.1 runs. It may be more due to the changes for EMMA than this issue.

161. **myrna:** Indeed, those errors were related to DERBY-3445; apologies for the confusion.

162. **mzaun:** With the remaining work on System Pribileges partitioned into the subtasks DERBY-3476, DERBY-3477, DERBY-3482, DERBY-3488, DERBY-3491, and DERBY-3495, I'm unassigning myself from this JIRA and will next look into DERBY-3488 (functional tests for system privileges).

163. **djd:** Kathey's work in DERBY-3534 is showing the impact of this change (the network authentication part) on existing applications.

164. **mzaun:** While NetworkServerControl authentication was a missing security feature that 2109 attempted to address, there were a couple of unresolved questions at the time of the decision to postpone some of the 2109 until after 10.4, for instance:

    - If I remember correctly, there was a discussion how NSC authentication relates to JMX, which has its own authentication mechanism; a question was if we then require two authentications, one by JMX and another one by NSC, or if NSC in this case does not do its own authentication check for shutdown. But I'm not sure remember this topic correctly.

    - Should we require user credentials for network server startup when running with authentication or should NSC not check user credentials upon shutdown when the server was started without ones? (Or should we have a default identity "APP" like with jdbc clients?)

    When browsing through the committed parts of the patch, it slipped my attention that the authentication check in NetworkServerControlImpl was still in effect (unlike the check

for authorization). If we decide to disable them for 10.4 it can be very easily done by making checkShutdownPrivileges() a dummy method (and keeping the remaining code for user credentials support).

165. **djd:** > ... how NSC authentication relates to JMX ...

    I think that's a future enhancement, it does not block DERBY-2109 from completing. It's the concept of a single-sign on, probably implemented through JAAS.

    > Should we require user credentials for network server startup when running with authentication or should NSC not check user credentials upon shutdown when the server was started without ones?

    I think those are two independent questions, rather than an "or" situation. For the second, if there is no system authentication then it should be transparent to the shutdown mechanism since Derby provides a no-op authentication in that case. So the user credentials are always checked but will always succeed. I think that's what is intended with the current implementation.

    > the authentication check in NetworkServerControlImpl was still in effect

    That was intentional on my part, the patch contained what I believed to be working code related to network server authentication so I committed it. The authorization had issues related to J2ME so I did not commit it. I was assuming work would continue incrementally to add authorization.

166. **kmarsden:** If DERBY-2109 has been postponed, should we back out theuser visible changes affecting existing applications on the 10.4 branch? Based on myshort experience, I think that the changes are going to be painful for Network Server users.
    If we are going to be making more changes for startup etc, I would
    rather hit them only hit users once with all the changes once they are
    ready for 10.5.

167. **djd:** Guess it depends on the definition of painful? Be careful of using the tests as an indicator of a typical application.

    The required change, as far as I understand, is that an application that is using system level authentication now needs to create its NetworkServerControl objects with a user name and password, rather than with none. E.g. they need to replace:

    NetworkServerControl ctrl = new NetworkServerControl();

    with

    NetworkServerControl ctrl = new NetworkServerControl(user, password);

    Or if they are using the command line, they need to add arguments
    -user user -password password

    I believe applications that do not use system authentication can continue to use the NetworkServerControl constructors that do not take a user name & password.

So are those two steps "painful" for an application picking up a new release of its database engine which will close a security hole?

168. **rhillegas:** Could someone crisply state the compatibility implications of what is currently checked in? I think that SystemPrivilegesBehaviour.html describes potential 10.5 issues.

169. **kmarsden:** I think it is more painful than just that because users now have tomanage the password, adding some sort of ui to accept the password andpossibly making scripts that could once run unattended now need anattendant. If I understand correctly, in a later release they willalso need to change their policy file and start will also beimpacted. I am not saying we shouldn't make these changes forsecurity's sake, just that we should group them together so that wedon't introduce new incompatibilities with every new release.

    Kathey

170. **mzaun:** Not much to add to the description by Dan of the user-visible changes with NetworkServer shutdown authentication, this is pretty much it. But to restate and summarize:

    1) If running derby server without authentication: no backward compatibility issues. (The old command-line usage, APIs, and derby tests continue to work.)

    2) If running a derby server with authentication on:
    a) The command-line usage of NetworkServerControl to shutdown the sever changes:
    java org.apache.derby.drda.NetworkServerControl shutdown
    results in an exception:
    08004:Connection authentication failure occurred. Reason: Invalid authentication..

    The remedy is to provide credentials:
    java org.apache.derby.drda.NetworkServerControl shutdown -user <user> -password <password>

    b) The NetworkServerControl API usage to programatically shutdown a server changes:
    NetworkServerControl nsc = new NetworkServerControl();
    nsc.shutdown();
    results in a java.sql.SQLException: Connection authentication failure occurred. Reason: Invalid authentication..

    The remedy is provide credentials to the NetworkServerControl constructor:
    NetworkServerControl nsc = new NetworkServerControl("user", "password");
    nsc.shutdown();

    c) Note that there is an edge case of b), which the current implementation only addresses poorly:
    NetworkServerControl nsc = new NetworkServerControl();
    nsc.start(console);
    ...
    nsc.shutdown();
    currently fails with above's SQLException. This is because of an asymmetrie we currently have: we don't require credentials when bringing up a server but we do require them when shutting down.

There is an easy workaround, however:
NetworkServerControl nsc = new NetworkServerControl();
nsc.start(console);
...
NetworkServerControl nscauth = new NetworkServerControl("user", "password"); nscauth.shutdown();

d) If users have their own tests that use derby's junit test framework, they'll have to use a test decorator that takes user credential arguments.

e) I'm not sure if users can programatically use derby's NetworkServerControlImpl (!) class to shutdown the server, but if so, they'll have to use an instance that has been constructed with user credential arguments, analog to b).

One more item is a not a backward compatibility issue: When running with authentication on, I think the server already requires user credential in connection URLs, for example:
connect 'jdbc:derby://localhost:1527/;shutdown=true';
fails but
connect 'jdbc:derby://localhost:1527/;shutdown=true;user=user;password=password';
succeeds in 10.3 already.

I don't have a strong opinion whether the shutdown authentication feature should be bundled with shutdown authorization checks later or not, but it would be nice to have a better solution for 2c).

171. **djd:** Just to add I think the current commit changes fix two bugs:

1) If a network server shutdown also shuts down the database engine and authentication is enabled then previously the database shutdown would fail. E.g. a clean shutdown would not occur and recovery would be needed.

2) Any user on the same machine could shutdown any network server even if they did not posses any valid authentication credentials.

172. **rhillegas:** Thanks to Martin and Dan for the crisp summary of the existing compatibility issues. I think that that we have fixed a very serious security problem. We should keep this more secure behavior in the 10.4 release even though, as Kathey points out, the extra security will cause extra pain.

I think this is pretty much how the industry responds to serious security breaches today. The trend is to push security patches to the field as soon as the patches are available and not defer them for a half year.

173. **djd:** What I should have said was the current commit *attempts to* fix two bugs:

Given DERBY-3537 and the behaviour seen in [1] I'm not convinced that it does fix this:

> 2) Any user on the same machine could shutdown any network server even if they did not posses any valid authentication credentials.

[1]
https://issues.apache.org/jira/browse/DERBY-3534?focusedCommentId=12578892#action_12578892

[edit to refer to correct problem]

174. **kmarsden:** The issue reported with:
https://issues.apache.org/jira/browse/DERBY-3534?focusedCommentId=12578892#action_12578892
where I thought the server was not coming down was user error. The test was bringing the server backup quickly and I thought it wasn't going down.

175. **kmarsden:** Will there be documentation updates and a release note for this issue?
It seems like it would make sense to file a separate issue for the work that
has been done and submit doc changes and releaseNote with that, then itcan be resolved and show up in the 10.4 release notes.

176. **mzaun:** > Will there be documentation updates and a release note for this issue?
I'm working on a documentation update and a release note proposal for NetworkServerControl authentication.

177. **mzaun:** Created DERBY-3585 for Release Note and documentation update proposal.

Chapter 3

Connected issue DERBY-3541

3.1 Summary

quit; on ij with authentication enabled does not shutdown the derby modules that are loaded

3.2 Description

3.3 Attachments

1. derby.properties
2. PrintStackTrace.diff
3. WithoutAuthentication_StackTrace.txt

3.4 Comments

1. **narayanan:** I modified the derby codebase to print a stack trace (new Throwable().printStackTrace();) inthe TopService#shutdown() method (pls find patch attached.)

   I then tried getting a connection *with* authentication enabled (pls find derby.properties attached) with the following connection url

   ij> connect 'jdbc:derby:replicationdb2;create=true;user=oystein;password=pass';
   ij> quit;
   vn@vn-laptop:~/work/workspaces/PrintStackTrace/trials$

   There was no stack trace printed.

   I then tried a connection *without* authentication enabled

   There was a stack trace printed (pls find attached stack trace)

   ij> connect 'jdbc:derby:replicationdb1;create=true';
   ij> quit;
   java.lang.Throwable
   at org.apache.derby.impl.services.monitor.TopService.shutdown(TopService.java:327)
   at org.apache.derby.impl.services.monitor.BaseMonitor.shutdown(BaseMonitor.java:237)
   at org.apache.derby.impl.services.monitor.BaseMonitor.shutdown(BaseMonitor.java:203)

at org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:231)
at org.apache.derby.jdbc.AutoloadedDriver.connect(AutoloadedDriver.java:119)
at java.sql.DriverManager.getConnection(DriverManager.java:582)
at java.sql.DriverManager.getConnection(DriverManager.java:207)
at org.apache.derby.impl.tools.ij.utilMain.cleanupGo(utilMain.java:416)
at org.apache.derby.impl.tools.ij.utilMain.go(utilMain.java:249)
at org.apache.derby.impl.tools.ij.Main.go(Main.java:215)
at org.apache.derby.impl.tools.ij.Main.mainCore(Main.java:181)
at org.apache.derby.impl.tools.ij.Main.main(Main.java:73)
at org.apache.derby.tools.ij.main(ij.java:59)

..... and so on (attached is the full stack trace)

Not shutting down the modules when authentication is enabled I believe is not correct.

2. **narayanan:** Derby-3447 relies on the stop() on the MasterController module being called in order to shutdown the log shipper thread. Since stop() is not called in MasterController upon a quit; in ij when authentication is enabled it results in the log shipper thread being active and cause a hang after quit; is called.

3. **narayanan:** Upon calling quit in ij, ij tries to shutdown the connection. The code relevant to this
is found here

   org.apache.derby.impl.tools.ij.utilMain.cleanupGo(utilMain.java:416)

   here we do the following

   ```
   if (d!=null) { // do we have a driver running? shutdown on exit.
   try {
   DriverManager.getConnection("jdbc:derby:;shutdown=true");
   } catch (SQLException e) {
   // ignore the errors, they are expected.
   }
   }
   ```

   now with authentication enabled trying to take a connection to shutdown like done in the above code will result in an exception

   ij> connect 'jdbc:derby:;shutdown=true';
   ERROR 08004: Connection authentication failure occurred. Reason: Invalid authentication..

   This is the reason why upon a quit; in ij with authentication enabled and in the embedded mode the modules are never shutdown.

4. **narayanan:** We get a better picture of the above error if we replace the catch clause in

   org.apache.derby.impl.tools.ij.utilMain.cleanupGo(utilMain.java:416)

   ```
   if (d!=null) { // do we have a driver running? shutdown on exit.
   try {
   DriverManager.getConnection("jdbc:derby:;shutdown=true");
   } catch (SQLException e) {
   ```

// ignore the errors, they are expected.
}
}
with a e.printStackTrace()

I get the following everytime I try to take a connection and quit ij with
authentication enabled

ij version 10.5
ij> connect 'jdbc:derby:mydb;user=oystein;password=pass';
ij> quit;
java.sql.SQLNonTransientConnectionException: Connection authentication failure occurred.
Reason: Invalid authentication..
at org.apache.derby.impl.jdbc.SQLExceptionFactory40.getSQLException(SQLExceptionFactory40.java:76)
at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Util.java:87)
at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Util.java:93)
at org.apache.derby.impl.jdbc.Util.generateCsSQLException(Util.java:172)
at org.apache.derby.jdbc.InternalDriver.connect(InternalDriver.java:220)
at org.apache.derby.jdbc.AutoloadedDriver.connect(AutoloadedDriver.java:119)
at java.sql.DriverManager.getConnection(DriverManager.java:582)
at java.sql.DriverManager.getConnection(DriverManager.java:207)
at org.apache.derby.impl.tools.ij.utilMain.cleanupGo(utilMain.java:416)
at org.apache.derby.impl.tools.ij.utilMain.go(utilMain.java:249)
at org.apache.derby.impl.tools.ij.Main.go(Main.java:215)
at org.apache.derby.impl.tools.ij.Main.mainCore(Main.java:181)
at org.apache.derby.impl.tools.ij.Main.main(Main.java:73)
at org.apache.derby.tools.ij.main(ij.java:59)
Caused by: java.sql.SQLException: Connection authentication failure occurred. Reason:
Invalid authentication..
at org.apache.derby.impl.jdbc.SQLExceptionFactory.getSQLException(SQLExceptionFactory.java:45)
at org.apache.derby.impl.jdbc.SQLExceptionFactory40.wrapArgsForTransportAcrossDRDA(SQLExceptionFactory
at org.apache.derby.impl.jdbc.SQLExceptionFactory40.getSQLException(SQLExceptionFactory40.java:70)
... 13 more
vn@vn-laptop:~/work/workspaces/PrintStackTrace$

5. **narayanan:** I get an authentication exception similar to the one above in a client/server environment too (with the same modification to the derby codebase as above).

   I however don't think I can fault derby here.
   I believe when we call quit; on ij (after booting a database in the embedded mode)we are trying to shutdown ij without worrying about the database that we mighthave booted earlier. I do not think we can restrict the user from exiting (shuttingdown) the application (ij) he has started just because he has taken an authenticatedderby connection inside the application(ij).
   This seems like a perfectly legal operation to me.

   ij relying on taking a successful system shutdown derby connection (without proper user name and password) when authentication is enabled causes the modules in the derby system to not be shutdown in a clean fashion.
   Basically ij tries its best to do a clean shutdown of the system upon exit.

   From the analysis above I am tempted to close this JIRA issue as invalid (unless someone
   has an objection).

I however think this behavior of ij when we call a quit (of not doing a clean shutdown when authentication
is enabled) can be documented somewhere (if not already documented). If the community feels the same
way I will raise a JIRA requesting this documentation to be included.

6. **jorgenlo:** I guess this (quiting derby without shutting down the databases) has other implications as well. E.g., when a booted database is not properly shut down, the checkpoint that is normally made on shutdown will not be there, increasing the time to recover the next time that database is booted.
   Combining what you say here with the discussion on 3447, my suggestion is twofold:

   * I agree that derby should not hang if the user shuts down an application without shutting down the databases. Your suggestion of using daemon threads for replication sounds good to me.
   * On the other hand, we control the ij application, and we may want to improve it. Documenting this problematic behavior like suggested may be enough, but it would also be possible to abort the 'quit;' call with an exception saying:

   ij > quit;
   Error: could not quit ij because the following databases have not been shut down: "test1", "test2". Shut down these databases first and then quit ij.

   I don't feel strongly about either option (only document / fail 'quit;'), but I think that in any case, the severity can be set lower now that we know the cause of the problem.

7. **narayanan:** >* On the other hand, we control the ij application, and we may want to improve it.
   >Documenting this problematic behavior like suggested may be enough, but it would>also be possible to abort the 'quit;' call with an exception saying:

   Depends on how much of derby specific code you want to put in ij, Jorgen.
   I must admit I was surprised at seeing the derby system shutdown being attempted
   when we do a quit itself.

   My impression about ij was ij is a general jdbc front end tool. It can be used with any database
   that supports a jdbc driver. This shutdown attempt with use to any other database (different
   jdbc url) will surely fail. Well maybe that is why we ignore the SQLExceptions upon doing this.

   Adding more and more derby specific code to ij, makes ij more and more a
   derby specific tool and lesser and lesser a general jdbc interface supporting front-end.

   For now I will lower the severity of the issue. I think we can take it from there.

   I will also leave the issue open.

8. **oysteing:** I think it is unfortunate that ij behaves differently with and without authentication.

   I also feel that is very convenient that ij shut down the database when I exit. Lately, when I have done some work in ij on a database with authentication/authorization turned

on, it has been very annoying that if I quit I ij after doing a large update operation, it has taken a long time to boot the database the next time. (Now I understand why)

I see the following possible solutions:

1. Change ij so the database is never shut down on quit, not just when authentication is turned on. Document that users need to shutdown databases before they exit ij, similarly to the message to JDBC programmers.
2. Somehow fix the case with authentication so that the database is shut down. (I am not sure how)
3. Let quit report error if ij is not able to shut down a database due to authentication. Introduce a new command that can be use to force quit ij without shutting down databases.

9. **espinha:** Triaged for 10.5.2. No changes made.

10. **lilywei:** I tried this bug on trunk version(10.6). I can not reproduce this bug.$ ij -p derby.properties
ij version 10.6
ij> connect 'jdbc:derby:test;create=true;user=johndoe;password=pass';
ij> quit;
I am welcome to any suggestion to reproduce this bug.

11. **narayanan:** Thank you for trying out the reproducible.when you said "I can not reproduce this bug."
did you mean that the modules are currently being
closed? Are you sure that TopService#shutdown()
is now being called with authentication enabled?

12. **lilywei:** Thank you, too. This is a good bug to look at. I mean that when authentication is enable, quit does not do shutdown. However, I don't see how quit can shutdown when authentication is enable. In such case, shutdown process will hit authentication failure error first before successfully shutting down the database. I think that is the point 2 on Oystin's point if I may quote him.

13. **narayanan:** Oystein says:

    "2. Somehow fix the case with authentication so that the database is shut down. (I am not sure how)"

    Although he does mean that it is difficult to shutdown the database when authentication is enabled
    (the same way shutdown is done when authentication is not enabled) he does not mean this is not
    a bug.

    Like in my comments before, ij is a generic jdbc front end tool, I am not happy with a derby specific
    call here at all. Ideally speaking you should be capable of using ij with any DB that supports a jdbc
    driver.

    If everyone agrees we can probably document that we need to shutdown DB's before quitting and
    prevent ij from going down without closing the DB's. That would be one probable solution.

But again
I am not sure I will be happy with such a crude fix.

Here is my 2 paisa :)

(IJ is a simple and elegant tool. There is a innocent simplicity to the tool that I would never want it
to lose. If a user wants a sophisticated front end there are other options. We should not clutter ij
unnecessarily.)

14. **kmarsden:** V. Narayanan said:
>If everyone agrees we can probably document that we need to shutdown DB's before quitting and prevent ij from going down without closing the DB's.
I think that is just too dangerous. There are a lot of applications that use ij scripts during their install that might hang if we do this. We do document that exit (quit) does a shutdown.
http://db.apache.org/derby/docs/dev/tools/rtoolsijcomref33358.html

I think the best we can do is document that with authentication, shutdown should be performed explicitly because the ij request will not succeed. I think even adding a warning that shutdown was not successful might cause automated scripts to fail.