

User Interface - Feature #4129

map web client users to OS accounts

07/03/2019 04:58 PM - Greg Shah

Status:	Closed	Start date:	
Priority:	Normal	Due date:	
Assignee:	Galya B	% Done:	100%
Category:		Estimated time:	0.00 hour
Target version:		version:	
billable:	No		
vendor_id:	GCD		
Description			
Related issues:			
Related to User Interface - Feature #3931: single sign-on for virtual desktop...			Closed
Related to User Interface - Feature #4517: optionally back the 4GL features f...			Closed

History

#1 - 07/03/2019 06:16 PM - Greg Shah

Existing Approaches

The web client is designed with a server-driven spawning process for launching the FWD client JVM. This means that the operating system user account must be known at that time, since the client JVM must run with that user's context/permissions.

1. In virtual desktop mode, we have a login prompt that completely delegates the userid/password to the end user.

2. Embedded Mode

The following is implemented in WebHandler.spawn() which is used to launch a new web client.

User	Password	Override Configured	Trusted Configured	Default Account Configured	Result
-					
non-null	non-null	no	n/a	n/a	Spawn using the given credentials.
non-null	non-null	yes	n/a	no	Error (override mode requires a default account).
non-null	non-null	yes	n/a	yes	Spawn using the default account and the given pw.
non-null	null	n/a	no	n/a	Error (pw can't be null unless trusted mode is allowed).
non-null	null	no	yes	n/a	Spawn using the given user.
non-null	null	yes	yes	no	Error (override mode requires a default account).
non-null	null	yes	yes	yes	Spawn using the default account.
null	non-null	no	n/a	no	Error (null user requires a default account).
null	non-null	yes	n/a	no	Error (override mode requires a default account).
null	null	no	n/a	no	Error (null user requires a default account).
null	null	yes	n/a	no	Error (override mode requires a default account).
null	null	yes	no	yes	Error (pw can't be null unless trusted mode is allowed).
null	null	yes	yes	yes	Spawn using the default account.

Trusted mode is attempted when the password is passed as null, but it will only work if the mode is also enabled in the server's configuration.

The referenced configuration is stored in the directory using webClient/override, webClient/defaultAccount. For trusted mode, the trustedspawner resource plugin must be enabled and the resources/ACLs implemented.

Using trusted mode, one can force the OS user to a specified value and the spawning can occur without a password (it also requires the spawner to be implemented with suitable admin/root rights). Without this, the caller must either provide both a userid and password OR a default account can be configured. This default account approach won't work for most multi-user systems since only 1 account name can be specified.

Improvements

1. OS User Pooling and Mapping

For security purposes, it may be important to isolate each user in a separate OS account. Such a thing must at least be possible. But doing this brings the added cost of managing the user accounts at the OS level. Today there are no built-in facilities in the FWD admin console to manage OS users, so this administration must be done using the OS.

Where possible, customers will want to use a pool of generic user accounts and allocate an OS account to a specific user for the duration of that user's application session. Multiple pools would be needed in the case where different application user roles required different OS permissions/access. Such an approach would assume that it would not matter which account is used from a pool for a given role. This would only work for applications that have no persistent user-level storage of any files (configuration, data or whatever).

We would need to be able to configure these pools, define the pool size and how which OS account names are included. We may even want to provide some option for a name spec that we can use to generate multiple names (e.g. user99 where 99 is a placeholder for a generated 2 digit number). Each pool would have a name, which applications might map to a user role.

Some mechanism for mapping application users to the OS pools would be needed. This would have to be done in coordination with an application login approach that is probably designed as an API.

2. Virtual Desktop Mode

Customers have requested that there be only a single login for both the operating system and the application (a.k.a. **single sign on**). For this to work, the OS account must be known before the application gets a chance to process, so using the standard application login does not fit well in this approach. This means modifying the flow of the login and moving to an API approach.

One possible solution is to provide hooks for the virtual desktop mode login dialog. Today that dialog is purely for the OS account. But instead it could be made more generic and hooks could be provided to call the application login API. That API would need to authenticate with the application and return:

- A specific OS userid (and optionally a password).
- A specification of the pool from which to allocate an OS user.

The spawning would then continue using that OS account. I think this means we would be using trusted spawning mode. The application itself would need to read the authenticated user or have that user set when the application starts (which can only happen after the application is launched).

One consideration would be to allow that login to integrate with the same `WebHandler.spawn()` API that is used by Embedded Mode. Thus, the same features enabled there can be used in virtual desktop mode as well.

3. Embedded Mode

Although the spawning capability already does provide some facilities to hide the OS account mapping, we do not provide any built-in facility in the standard embedded mode client for taking advantage of this.

Enhancing the `WebHandler.spawn()` API to include the above described pool support and mapping feature is a core idea here. The separate application could pass this information or it could be returned from the authentication API in the converted application.

4. Allocate/Deallocate Hooks

It is likely that some processing is needed in association with allocation or deallocation of pooled accounts. This might be needed to ensure that the account is setup with a known configuration or to cleanup state left behind by the application.

Allocation is the association of an application user to an account from the pool. It is a kind of check-out or removal from the pool for the lifetime of the application session.

Deallocation is when the application user's session ends, the application user is disassociated from the OS account and the account is returned to the pool.

In both cases we should allow the registration of a hook that gets called to process on these events. I think the code will need to process within the context of the OS account. The allocation hook must be called after the client JVM is launched in the OS account but before any application processing can occur. The deallocation hook would be called after all application processing is done, but before the client JVM exits.

This could be implemented as downcalls from the server that brackets the `StandardServer.standardEntry()` processing (virtual desktop mode) OR the equivalent in embedded mode. As with our authentication plugin support, we probably need to serialize the registered Java hook class and send it down to the client.

#2 - 08/22/2019 01:16 PM - Greg Shah

- Related to Feature #3931: single sign-on for virtual desktop mode added

#3 - 08/14/2020 08:45 AM - Greg Shah

- Related to Feature #4517: optionally back the 4GL features for Registry access with the user-specific offline storage features added

#4 - 08/14/2020 08:47 AM - Greg Shah

The mapping of end users to OS account users should be stable and if possible 1-to-1. The large advantage of this approach is that any persistently stored configuration (for example, through INI files or via the registry in [#4517](#)) will be available to the user which will match the legacy system.

#5 - 10/29/2021 09:32 AM - Greg Shah

We have a customer that wishes to run all users in a single OS account. They understand the security implications and the application itself is written to be tolerant of this case. We must ensure that this use case continues to be possible.

Most customers will probably not be able to do this.

#6 - 10/29/2021 09:52 AM - Greg Shah

I think we also need the option to map to the FWD account as well. Today we mostly use the "Guest" mode plugin and bogus account but that really doesn't give us many options from a security perspective. It seems like the mapping should handle both OS and FWD accounts from each application account identity.

#8 - 06/16/2023 04:20 AM - Galya B

What is FWD user? I understand the admin has access to the admin login and temp accounts are used for NativeSecureConnection authentication, but what do we achieve with bogus auth without password?

#9 - 06/16/2023 08:38 AM - Galya B

OS users will have their passwords in directory encrypted like for admin user. But I can't recall do we expect the app users password verification to be done in the FWD code, i.e. having their passwords also in directory?

#10 - 06/16/2023 09:17 AM - Greg Shah

What is FWD user? I understand the admin has access to the admin login and temp accounts are used for NativeSecureConnection authentication, but what do we achieve with bogus auth without password?

This account sets the context for any security decisions by the FWD SecurityManager and its registered plugins. Currently, we take a quick and dirty approach and we make all web clients use the same account. It is not a good approach but it does limit the set of web clients to only those resources allowed for bogus. What we want is to have the capability to set the FWD account to something more specific instead of all web clients sharing the same FWD account.

OS users will have their passwords in directory encrypted like for admin user.

I'm not sure this is needed in Linux/UNIX since the spawner is suid and can spawn without the password. In Windows, this might be a requirement but I'm not sure.

But I can't recall do we expect the app users password verification to be done in the FWD code, i.e. having their passwords also in directory?

No, the idea is to implement a callback to a registered authentication service that is provided by the application ([#3931](#), [#3770](#), [#4571](#)). If this call succeeds, then we take the identity it provides and we use that to map the OS and FWD accounts.

#12 - 06/20/2023 07:30 AM - Galya B

Before starting I need to have the directory.xml structure approved:

```
<node class="container" name="">
  <node class="container" name="security">
    <node class="container" name="accounts">
      <node class="container" name="ospools">
        <node class="ospool" name="default_pool">
          <node-attribute name="fwduser" value="bogus"/>
          <node-attribute name="osuserlist" value="osuser1,osuser2"/>
        </node>
        <node class="ospool" name="admin_pool">
          <node-attribute name="fwduser" value="admin"/>
          <node class="osuser" name="user3">
            <node-attribute name="password" value="9FfyMaC4lEXNP5bIUo5TRM4akjw="/>
          </node>
        </node>
      </node>
    <node class="container" name="appusers">
      <node class="app" name="app1">
        <node class="appuser" name="appuser1">
          <node-attribute name="ospool" value="admin_pool"/>
        </node>
      </node>
    </node>
  </node>
</node>
```

I have some questions:

1. Can I mix <node-attribute> and <node> as children? I believe there shouldn't be a limitation, but I haven't seen an example.
2. When using a name for the class instead of container does it imply some auto-mapping to pojo classes? I think it's not related to parsing the xml.
3. App users can be different sets for different apps. Do we expect multiple apps to be running in the same server?

#13 - 06/20/2023 05:02 PM - Greg Shah

Before starting I need to have the directory.xml structure approved:

It is not fully clear the purpose of the appusers container. I also don't see exactly how we would implement a one-to-one mapping, which is the most common case.

Please summarize the logic of how this will be used.

1. Can I mix <node-attribute> and <node> as children? I believe there shouldn't be a limitation, but I haven't seen an example.

I'm not sure.

2. When using a name for the class instead of container does it imply some auto-mapping to pojo classes? I think it's not related to parsing the xml.

Not exactly. You can define a data structure in the schema (see src/dir_schema.xml for the definitions).

3. App users can be different sets for different apps. Do we expect multiple apps to be running in the same server?

Yes.

#14 - 06/21/2023 06:31 AM - Galya B

Greg Shah wrote:

I also don't see exactly how we would implement a one-to-one mapping, which is the most common case.

In the example it can be achieved by creating a pool with one os user and assigning it to an app user.

If we cleanup after each user, do we really need 1:1?

It is not fully clear the purpose of the appusers container.

OS user pools and app users should be in separate containers. If we have app user container for each app, then app user containers can be on a level higher without the wrapper appusers.

I'm not sure though how the app container can be matched to the running app?

Please summarize the logic of how this will be used.

- Adding a new container under accounts for listing os users (with and without password).
- Adding a new container(s) under accounts for listing in-app users for each app. All of them can be wrapped by one top container as in the example.
- Mapping of OS user pools to in-app users is done by specifying the OS user pool for each in-app user.

#15 - 06/22/2023 03:57 PM - Greg Shah

If we cleanup after each user, do we really need 1:1?

I'm not sure what you mean by cleanup, but I suspect the answer is yes we really need 1:1. Customers using this 1:1 mode will have long lived users which potentially have state/config/data/documents stored in the user's home dir. We don't want to "cleanup" that content or touch it in any way after some possible initialization step.

Please consider that we:

1. Need to associate each active user with a FWD account.
2. We already have a FWD account structure in the directory.

It would be a natural approach to store some amount of this mapping information in the FWD account, if possible.

OS user pools and app users should be in separate containers. If we have app user container for each app, then app user containers can be on a level higher without the wrapper appusers.

Do we need to store information about the app users? I was thinking that was a matter for the application authentication hook. If the auth hook itself accepts the app userid and password, authenticates it and then just returns the FWD account, we can potentially store the mapping in the FWD account. Then we could just read the OS user that was referenced there and lookup the OS user container to get the password (if needed). In suid

cases, we wouldn't even need any other container at all.

#16 - 06/23/2023 01:30 AM - Galya B

Greg Shah wrote:

It would be a natural approach to store some amount of this mapping information in the FWD account, if possible.

I designed it the way normalization in databases is designed to allow maximum flexibility in mapping. One and the same app user can have different permissions in different applications. App users need to be top level model/container in my opinion instead of spreading them all over another container and trying to collect them in an in-memory structure to be easily searched through.

When a login attempt for certain user succeeds, we can easily directly query directory knowing exactly the path for the app user and get the mapping for the FWD user and OS user (pool) and directly collect these data instead of keeping all of these in-memory. This will also allow for easy dynamic update of the data runtime.

Do we need to store information about the app users?

No, nothing other than knowing what FWD user and OS user is mapped to the app user and what application it belongs to.

#17 - 06/23/2023 01:43 AM - Galya B

Greg Shah wrote:

If we cleanup after each user, do we really need 1:1?

I'm not sure what you mean by cleanup, but I suspect the answer is yes we really need 1:1. Customers using this 1:1 mode will have long lived users which potentially have state/config/data/documents stored in the user's home dir. We don't want to "cleanup" that content or touch it in any way after some possible initialization step.

I was left with the impression that one-to-many will require cleanup to restore the system in the state before running the app, so that the next app-user reusing the os-user will not be able to access sensitive data from the previous one. Now I understand we also need a mode where there is no cleanup. This means we should provide a way to explicitly configure if the system is running with 1:1 or 1:many mapping. Do you think mixing both in runtime should be an option?

#18 - 06/23/2023 06:19 AM - Greg Shah

Do we need to store information about the app users?

No, nothing other than knowing what FWD user and OS user is mapped to the app user and what application it belongs to.

Right, so let's not store them there. Otherwise it is something else for the admin to maintain in addition to the fact that the application will already have its own database of app users/passwords. I don't want to duplicate this when we don't actually need that information. We can let the API/Hook (implemented on a custom basis by the application) provide us the needed information.

This needed information might be as simple as returning back the FWD account that is to be associated with the session. If the session also entails spawning, then whatever OS user is defined in that account could be looked up and used. This takes advantage of our existing admin screens, minimizes the additional data that is needed to be stored and still provides the core feature.

This means we should provide a way to explicitly configure if the system is running with 1:1 or 1:many mapping. Do you think mixing both in runtime should be an option?

Yes, I do think that some users might need to have a dedicated account and other users can have a shared account. The thing is that even the shared account case might not need any cleanup. Some applications are written to allow this to work (one of the apps we are dealing with is already that way).

I think the setup/cleanup hooks are optional and perhaps these should be aspects of the API.

#19 - 06/23/2023 10:02 AM - Galya B

Is the argument about in-memory data structure not valid?

#20 - 06/23/2023 10:05 AM - Greg Shah

Is the argument about in-memory data structure not valid?

Not that I know of. We have no plan to store things in-memory.

#21 - 06/23/2023 10:08 AM - Galya B

Greg Shah wrote:

Not that I know of. We have no plan to store things in-memory.

How do you find the mapping of the app user when the auth hook returns success? Loop through all fwd users?

#22 - 06/23/2023 10:30 AM - Galya B

Greg Shah wrote:

We have no plan to store things in-memory.

If there are any plans I don't know of, please share the directory.xml structure with me and how it's supposed to work, so that I can start implementing it. Otherwise I'm trying to guess some intentions and I don't seem to be good at it.

#23 - 06/23/2023 10:38 AM - Greg Shah

Not that I know of. We have no plan to store things in-memory.

How do you find the mapping of the app user when the auth hook returns success? Loop through all fwd users?

We don't need the app user. The hook can return the FWD user that corresponds to the app user that was authenticated. Then we look that up FWD account and use the details contained there.

#24 - 06/23/2023 10:39 AM - Greg Shah

We have no plan to store things in-memory.

If there are any plans I don't know of, please share the directory.xml structure with me and how it's supposed to work, so that I can start implementing it. Otherwise I'm trying to guess some intentions and I don't seem to be good at it.

I have no hidden plans. I may have forgotten to mention something but at this point in time, as far as I know, everything is documented.

#25 - 06/23/2023 10:47 AM - Galya B

Greg Shah wrote:

We don't need the app user. The hook can return the FWD user that corresponds to the app user that was authenticated. Then we look that up FWD account and use the details contained there.

I have no hidden plans. I may have forgotten to mention something but at this point in time, as far as I know, everything is documented.

I thought the task is to map app-users to os-users in directory.xml, so this is somewhat different from what I got. Basically the task should be to map fwd-users to ... something?

If the customer's app will do the fwd user : app user mapping, we don't have to know anything about anything really. Let them create their interface and deal with the mapping and ask them to return the os user as well... I mean they will have to do a mapping to fwd user anyways as far as I understand.

#26 - 06/23/2023 11:37 AM - Greg Shah

I thought the task is to map app-users to os-users in directory.xml, so this is somewhat different from what I got. Basically the task should be to map fwd-users to ... something?

In the task title I used "web client users" which is meant to be a reference to the end user of the system. That user is identified by a tuple of app user (managed by the application and not FWD), the FWD account which is used for the context of the FWD session and the OS user which is required for web client spawning and defines the OS level context of the FWD client process.

If we assume that FWD really doesn't care about the app user AND if we further assume that the application can handle the mapping to a FWD account, then we just need to handle the mapping from a FWD account to an OS account. I'd like to do that with limited affect on the existing data structures and admin screens.

For an OS account to be shared across more than one FWD account, we would need for the OS account info to be stored separately from the FWD accounts. Since we already store FWD accounts, we can add additional information to that, such as the mapping of the OS account. That would mean that we would need to have a new admin screen for adding/editing/deleting/viewing the OS level account definitions IF this is needed. For the common case in Linux, we may not need to define a separate OS account definition because we use an suid spawner that doesn't need the OS account password. If the OS account name is already stored as a mapping in the FWD account AND we don't need a password then we probably don't need to otherwise maintain a separate store/list of the OS accounts.

It may be that we don't really need a pool concept because the mapping of a given FWD account to OS account can naturally be either 1:1 or 1:many and it doesn't really need to be defined in advance. I don't think we even really need to know from a FWD perspective. That simplifies things I think.

We need to consider the initial allocate/deallocate or other hooks that may be needed. My concept from above might not be the correct model. I'd like you to speak with Roger and also with a customer to assess the needs in this area. I will send details about that via email. Because we require the OS account for spawning, it brings a requirement that these accounts be created/deleted, possibly initialized, passwords maintained. Either the customer will have to write code for that (each one duplicating the same functionality) or perhaps we should handle it ourselves with some OS-level integration.

#28 - 06/26/2023 04:23 AM - Galya B

Greg Shah wrote:

If we assume that FWD really doesn't care about the app user

the mapping of a given FWD account to OS account can naturally be either 1:1 or 1:many and it doesn't really need to be defined in advance. I don't think we even really need to know from a FWD perspective.

Server-side offline storage for client preferences requires unique identifier of the user that is consistent between sessions and browser updates. If this can't be app user or fwd user, I can't think of a way to reliably offer the functionality client preferences.

#29 - 06/26/2023 09:23 AM - Galya B

Just throwing a random idea here: If on successful login the app returns a unique id for the user instead of username, that will do the same in terms of identifying a unique user without giving out any other details. The id can be generated randomly explicitly for fwd, but it should be then stored in the app db or it can be the username encrypted with a salt value, so that it can be replicated the same on each login. Anyways that adds up to the complexity of maintaining the app in relationship to FWD.

Another related subject is the fact that client logs currently can't be matched to a user after the session is terminated. Actually is the pid is stored in the app db on session start, this may help somewhat before machine restarts.

#30 - 06/26/2023 09:42 AM - Galya B

Adding a few more details to the random idea above: Instead of creating multiple fwd users in the admin panel with the same characteristics, the admin can create just a few that act as groups. On successful login the unique identifier for the app user can be attached to the fwd name and be returned, like bogus28146, admin4574.

#31 - 06/26/2023 11:28 AM - Greg Shah

Galya B wrote:

Greg Shah wrote:

If we assume that FWD really doesn't care about the app user

the mapping of a given FWD account to OS account can naturally be either 1:1 or 1:many and it doesn't really need to be defined in advance. I don't think we even really need to know from a FWD perspective.

Server-side offline storage for client preferences requires unique identifier of the user that is consistent between sessions and browser updates. If this can't be app user or fwd user, I can't think of a way to reliably offer the functionality client preferences.

The FWD user is unique enough. If customers are sharing the account, then they either must accept that any stored state (server or client) is going to be available to all of the users that share the account OR they will need to ensure that state is not stored. This is an acceptable limitation. I don't want to add an extra layer of indirection.

#32 - 06/26/2023 11:31 AM - Greg Shah

Just throwing a random idea here: If on successful login the app returns a unique id for the user instead of username, that will do the same in terms of identifying a unique user without giving out any other details. The id can be generated randomly explicitly for fwd, but it should be then stored in the app db or it can be the username encrypted with a salt value, so that it can be replicated the same on each login. Anyways that adds up to the complexity of maintaining the app in relationship to FWD.

I prefer to avoid this added complexity.

Another related subject is the fact that client logs currently can't be matched to a user after the session is terminated. Actually is the pid is stored in the app db on session start, this may help somewhat before machine restarts.

We should consider the idea of writing some limited server state to the client log so that the server session can be matched.

#33 - 06/26/2023 12:07 PM - Galya B

Greg Shah wrote:

The FWD user is unique enough.

I can't imagine a customer who will create FWD users for each of their app / end users.

If customers are sharing the account, then they either must accept that any stored state (server or client) is going to be available to all of the users that share the account OR they will need to ensure that state is not stored. This is an acceptable limitation.

OK, I get it, FWD won't bother about uniqueness of end users and who actually shares anything available to that FWD user.

#34 - 06/26/2023 01:29 PM - Greg Shah

The FWD user is unique enough.

I can't imagine a customer who will create FWD users for each of their app / end users.

We have multiple customers doing this already. It is in fact the natural approach for any application which was written with the concept that each user had a unique account. Many applications have this assumption, especially ones that started as Windows GUI applications.

#35 - 06/26/2023 03:22 PM - Roger Borrello

1. Should we implement a facility in our admin UI to create/delete and manage OS accounts? This probably includes some ability to manage the password.

If this is intended to get rid of the need for applications to include "add user" function, that would be a big YES. It would keep developers from having to write a lot of code to do some simple stuff.

2. What hooks do we need for initialization or cleanup of the account. For example, if an applications needs some files and a directory structure placed in the home dir for a newly created account.

There are built-in hooks already in the Linux OS to control other aspects of a user's configuration. See `useradd -D` output. The config file is `/etc/default/useradd`.

```
rfb@rfb:~/projects/fwd/4938a$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

If you want a `/var/spool/mail/$USER` directory created for mail, the config would change to match the requirements. The above example is from my laptop.

Those hooks might need to be overridden when creating users, including specifying the group membership.

For deletion, one requirement I have seen is to check on the group membership before deleting, to make sure only "standard" user (ids between 500 and about 65534) are removed. The act of deleting should also remove any created mail spooler and home directory.

As for files/directories, I can see where initial `user_client.xml` might be needed to be put in place if that is configured in the `clientConfig` of the directory. As an aside, I just checked, and the spawner fails when that file doesn't exist. That might need to be addressed, perhaps ignoring if it configured and doesn't exist? Or should that be an error?

#36 - 06/26/2023 03:37 PM - Greg Shah

FWD doesn't know anything about user_client.xml. What are you referencing here?

#37 - 06/26/2023 03:49 PM - Roger Borrello

Greg Shah wrote:

FWD doesn't know anything about user_client.xml. What are you referencing here?

True. But the question was along the lines of, "What hooks do we need to position user files when a user is created?". One answer would be one that can build user_client.xml files, if needed.

#38 - 06/29/2023 08:17 AM - Galya B

Is config/auth-plugins/option the way of defining FWD user for the web process? Its value is bogus in testcases, but that option node is quite vague if that's what I'm looking for.

#39 - 06/29/2023 08:23 AM - Galya B

Also the FWD user bogus is listed in subjects in multiple security/acl resources and I don't see a way to update the list in the admin ui. If all app users are mapped to fwd users 1:1, I guess they will have to be added to the same subjects lists when accessing virtual desktop like for example com.goldencode.p2j.main.Spawner. Is this supposed to be done by hand in directory.xml?

#40 - 06/29/2023 08:39 AM - Greg Shah

Add/remove/edit users and groups is in the admin UI. ACL add/remove/edit is also in the admin UI. Groups are subjects too and ACLs can be specified based on groups.

#41 - 06/29/2023 08:43 AM - Greg Shah

Is config/auth-plugins/option the way of defining FWD user for the web process? Its value is bogus in testcases, but that option node is quite vague if that's what I'm looking for.

See SecurityCache.readAuthPlugins(). The option value has a plugin-specific meaning. For the GuestAccess plugin, the option is the FWD account to use for guest access.

#42 - 06/30/2023 03:29 AM - Galya B

Greg Shah wrote:

1. Should we implement a facility in our admin UI to create/delete and manage OS accounts? This probably includes some ability to manage the password.

The downside of such facility is it can't prepare remote brokers. As far as I understand all brokers should support the same set of OS accounts.

#43 - 06/30/2023 04:43 AM - Galya B

- File Screenshot from 2023-06-30 11-39-49.png added

Greg Shah wrote:

Add/remove/edit users and groups is in the admin UI. ACL add/remove/edit is also in the admin UI. Groups are subjects too and ACLs can be specified based on groups.

I think this (attaching a screenshot) should be the UI you refer to, but it's all greyed out.

#44 - 06/30/2023 05:04 AM - Galya B

Greg Shah wrote:

I can't imagine a customer who will create FWD users for each of their app / end users.

We have multiple customers doing this already. It is in fact the natural approach for any application which was written with the concept that each user had a unique account. Many applications have this assumption, especially ones that started as Windows GUI applications.

Currently how is mapping Windows GUI app users to fwd users 1:1 done?

#45 - 06/30/2023 06:36 AM - Greg Shah

1. Should we implement a facility in our admin UI to create/delete and manage OS accounts? This probably includes some ability to manage the password.

The downside of such facility is it can't prepare remote brokers.

Good point. The remote broker interface would need enhancement to provide account management features.

As far as I understand all brokers should support the same set of OS accounts.

This is "undefined". We added the broker capability long after the original design was determined. We did not consider this aspect. We might need to define one or more possible modes. OS accounts duplicated is one option. OS account stickiness/affinity is another option. The latter is probably the more obvious approach since most of these apps do expect that there is some persistence to the OS account between application startups.

#46 - 06/30/2023 06:37 AM - Greg Shah

Galya B wrote:

Greg Shah wrote:

Add/remove/edit users and groups is in the admin UI. ACL add/remove/edit is also in the admin UI. Groups are subjects too and ACLs can be specified based on groups.

I think this (attaching a screenshot) should be the UI you refer to, but it's all greyed out.

The management comes from the Accounts and Access Control menus.

#47 - 06/30/2023 06:40 AM - Greg Shah

I can't imagine a customer who will create FWD users for each of their app / end users.

We have multiple customers doing this already. It is in fact the natural approach for any application which was written with the concept that each user had a unique account. Many applications have this assumption, especially ones that started as Windows GUI applications.

Currently how is mapping Windows GUI app users to fwd users 1:1 done?

Largely the same way as for Linux. The exception is that the spawning works a little differently on Windows due to OS API differences. Running FWD as a Windows Service, using an administrator account and the requirement to have the OS account's password are all differences if I recall correctly. Eugenie will recall this better than myself.

#48 - 06/30/2023 07:08 AM - Galya B

Greg Shah wrote:

The remote broker interface would need enhancement to provide account management features.

It will also require spawning a process on the broker with admin privileges.

OS account stickiness/affinity is another option.

It will require a different mechanism of allocating brokers to clients. Multiple clients may have to be executed on the same broker at the same time, while another broker is not used. This will be mixing infrastructure with app logic which is not ideal for scaling up.

We might need to define one or more possible modes.

I think we should decide on only one, otherwise certain features built on top of the certain mode will not work under the other one.

The latter is probably the more obvious approach since most of these apps do expect that there is some persistence to the OS account between application startups.

This can be done server-side stored under the fwd user name.

Forgetting about localStorage seems to me like the cleanest solution. Otherwise there is either host affinity to signed-in users, or reverse proxy as solution. Both of them making the framework with stiff infrastructure.

#49 - 06/30/2023 07:45 AM - Galya B

On the other hand I don't know how applicable is the case of the app storing user specific data on the broker that needs to be persisted between sessions. If this data can be isolated, it can be moved between hosts and restored, but 4GL apps seem quite intertwined with the OS to reason about it.

#50 - 06/30/2023 08:38 AM - Greg Shah

The remote broker interface would need enhancement to provide account management features.

It will also require spawning a process on the broker with admin privileges.

I'm not sure we have to spawn a new privileged process because the current broker/spawner is already an suid/admin process so that OS calls can be made from inside the broker.

OS account stickiness/affinity is another option.

It will require a different mechanism of allocating brokers to clients. Multiple clients may have to be executed on the same broker at the same time, while another broker is not used. This will be mixing infrastructure with app logic which is not ideal for scaling up.

Yes, but it may be required for applications to work properly. They would have to distribute their users manually based on expected usage.

I should note that our current round robin approach is not very smart either. Today it just spawns the next client on the next broker in the queue without regard to the load on the brokers.

The latter is probably the more obvious approach since most of these apps do expect that there is some persistence to the OS account between application startups.

This can be done server-side stored under the fwd user name.

I'm not talking about key/value storage. These are potentially dependencies in the OS file system itself, like data files, spreadsheets, reports or whatever. The fact that in the 4GL all interactive users pretty much always have a local file system that is stable and persistent across application restarts. Many of these applications have deep integration with the local platform. We must retain this.

Forgetting about localStorage seems to me like the cleanest solution. Otherwise there is either host affinity to signed-in users, or reverse proxy as solution. Both of them making the framework with stiff infrastructure.

I don't understand this point. The requirement is to allow a login to a particular web client to have an OS account with which it is associated. How does localStorage come into it?

#51 - 06/30/2023 08:40 AM - Greg Shah

On the other hand I don't know how applicable is the case of the app storing user specific data on the broker that needs to be persisted between sessions. If this data can be isolated, it can be moved between hosts and restored, but 4GL apps seem quite intertwined with the OS to reason about it.

Moving this stuff around is out of scope. We just need to ensure that the specific OS account that is configured is used for the given FWD account user.

#52 - 06/30/2023 08:47 AM - Galya B

Greg Shah wrote:

Moving this stuff around is out of scope.

Speaking of out of scope, what do you say about revamp of brokers under "map users to OS accounts"?

#53 - 06/30/2023 08:47 AM - Galya B

Greg Shah wrote:

I don't understand this point. The requirement is to allow a login to a particular web client to have an OS account with which it is associated. How does localStorage come into it?

Host affinity is related.

#54 - 06/30/2023 08:56 AM - Greg Shah

Moving this stuff around is out of scope.

Speaking of out of scope, what do you say about revamp of brokers under "map users to OS accounts"?

We need to decide how to handle the multiple brokers case. I think some solution there will be needed.

I don't understand this point. The requirement is to allow a login to a particular web client to have an OS account with which it is associated. How does localStorage come into it?

Host affinity is related.

We aren't talking about host affinity here. We are talking about knowing on which broker the OS account exists.

#55 - 06/30/2023 12:45 PM - Galya B

I've followed the process of creating a web client and documented in [Sign-in](#), but I can't figure out when is the fwd user associated with the security context created on the server. In StandardServer#standardEntry it already exists, so I guess it's part of registerWebClientSession, but in WebClientConfig I see only one user and that is the OS user.

#56 - 07/03/2023 06:03 AM - Galya B

I was finally able to follow the security context creation flow and most importantly the association with fwd user for web processes. The server sends the credentials to the client, then the client returns them and auth is successful.

At the moment there is only one auth-mode configured in directory.xml in our test project as well as in a customer's project. It configures GuestAccess and that is used for all clients but appserver as far as I can tell.

My question is if there can be different auth modes for different processes or it is only one for all?
If a new Authenticator for Virtual Desktop is introduced does it have to be integrated with all clients?

#57 - 07/03/2023 09:24 AM - Galya B

What about:

```
<node class="container" name="">
  <node class="container" name="security">
    <node class="container" name="accounts">
      <node class="container" name="osusers">
        <node class="container" name="default">
          <node-attribute name="osbogus" value="pass"/>
          <node-attribute name="osbogus1" value="pass1"/>
          <node-attribute name="osbogus2" value="pass2"/>
        </node>
        <node-attribute name="os-admin" value="adminpass"/>
      </node>
    </node>
  </node>
  <node class="container" name="users">
```

```
<node class="user" name="bogus">
  <node-attribute name="osgroup" value="default"/>
</node>
<node class="user" name="admin">
  <node-attribute name="osuser" value="os-admin"/>
</node>
</node>
</node>
</node>
</node>
```

FWD user can be mapped to either osgroup or osuser. When both attr are present, osuser will be used. If no attr is used, osusers/default pool will be queried if present. All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.

#58 - 07/03/2023 10:15 AM - Galya B

Currently embedded is using only trustedspawner and only one account webClient/defaultAccount, that's how it's hard-coded. To know the os user before spawning the client, that's the same case as with Virtual Desktop.

Do we want to have the same sso login page before both clients when sso enabled?

#59 - 07/03/2023 10:49 AM - Greg Shah

Currently embedded is using only trustedspawner and only one account webClient/defaultAccount, that's how it's hard-coded. To know the os user before spawning the client, that's the same case as with Virtual Desktop.

Do we want to have the same sso login page before both clients when sso enabled?

Probably not since the UI requirements might be different, but we will want the same backend auth/spawning process as much as possible.

#60 - 07/03/2023 10:50 AM - Galya B

Greg Shah wrote:

Probably not since the UI requirements might be different, but we will want the same backend auth/spawning process as much as possible.

It will be a very different flow. OS user and FWD user are already determined at the point the in-app login appears.

#61 - 07/03/2023 10:51 AM - Greg Shah

In both scenarios there will not be an in-app login.

#62 - 07/03/2023 10:52 AM - Galya B

In hotel gui isn't the embedded login an in-app login?

#63 - 07/03/2023 10:57 AM - Galya B

I mean for embedded there is no FWD login. We'll rework the one for Virtual Desktop, but for embedded there is nothing in place before running the app, basically no way to get any user input and determine os / fwd user.

#64 - 07/03/2023 11:01 AM - Greg Shah

In hotel gui isn't the embedded login an in-app login?

Not exactly. It is currently a custom integration to handle the login. In other words, it does not run the existing 4GL login UI.

In #3770 we intend to make this a more architected solution that should share the same auth backend as virtual desktop mode.

I mean for embedded there is no FWD login.

Yes, but that is only because currently we have a workaround in place.

We'll rework the one for Virtual Desktop, but for embedded there is nothing in place before running the app, basically no way to get any user input and determine os / fwd user.

As noted above, we do have an integration that "up calls" from javascript to do an authentication. This would be replaced by the same auth backend.

#65 - 07/03/2023 11:05 AM - Galya B

Reusing the auth back-end will make in-app users authentication consistent between both web drivers, but it will not improve on using only one os user and trustedspawner for embedded, which is in #4129#Improvements point 3.

#66 - 07/03/2023 11:07 AM - Galya B

As it is now, there is no user interaction / input before spawning the client process under the os user.

#67 - 07/03/2023 11:09 AM - Galya B

From #3770:

add 4GL API for authentication

call this API before spawning

How do you know the user/pass at that time?

#68 - 07/03/2023 11:13 AM - Greg Shah

but it will not improve on using only one os user and trustedspawner for embedded, which is in #4129#Improvements point 3.

I don't think we have a requirement that forces using only one os user for embedded mode. As far as the trustedspawner is concerned, I was assuming it is optional, but that it is likely we would want to use it for BOTH embedded and virtual desktop mode.

#69 - 07/03/2023 11:15 AM - Greg Shah

As it is now, there is no user interaction / input before spawning the client process under the os user.

There is no requirement that it work this way in the future. Since the designed of the SSO approach is already meant to be the same conceptual idea (a login to the app outside of the 4GL and then automatically spawn) for both embedded and virtual desktop modes, then it seems like we can have a somewhat common approach.

#70 - 07/03/2023 11:16 AM - Greg Shah

From #3770:

add 4GL API for authentication

I don't plan to keep this part. There is no need for the 4GL code to be called. This is the same idea as for virtual desktop mode.

#71 - 07/10/2023 04:33 PM - Greg Shah

My question is if there can be different auth modes for different processes or it is only one for all?

Yes, it should be possible to use different authentication approaches simultaneously. For example, I would expect we would continue to support our certificate based authentication for batch processes will also supporting this new API.

If a new Authenticator for Virtual Desktop is introduced does it have to be integrated with all clients?

I'm not thinking about this as an "Authenticator for Virtual Desktop". I'm really thinking about this as "an API which can be used by customers to implement custom authentication for any interactive client". Thus, it would be suitable for implementing across all interactive clients.

In regard to the directory structure, I'm OK iwth the /security/accounts/osusers/ section. In regard to the parts in /security/accounts/users/, are you proposing extending our existing accounts structure? That is my expectation.

All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.

Isn't our PAM support optional?

Greg Shah wrote:

If a new Authenticator for Virtual Desktop is introduced does it have to be integrated with all clients?

I'm not thinking about this as an "Authenticator for Virtual Desktop". I'm really thinking about this as "an API which can be used by customers to implement custom authentication for any interactive client". Thus, it would be suitable for implementing across all interactive clients.

`com.goldencode.p2j.security.Authenticator` is the interface for rpc connections auth configured in `auth-mode/plugin`, usually being `GuestAccess`. I've started working on the "API" in [#3931](#) with `SsoAuthenticator`, but this is indeed a different one and doesn't have much to do with the connection authenticator (`com.goldencode.p2j.security.Authenticator`). So here I speak about `com.goldencode.p2j.security.Authenticator` and the fact we need a new implementation that works in combination with the new "API". The connection auth should know how to get the fwd user associated with the currently logged-in app user and that is new. Directory configs currently allow only for one `auth-mode/plugin` to be specified for the server, so I'm still not sure how I'll solve that.

In regard to the directory structure, I'm OK with the `/security/accounts/osusers/` section. In regard to the parts in `/security/accounts/users/`, are you proposing extending our existing accounts structure? That is my expectation.

Yes, extending `accounts/users` with `osuser` and `ospool` attributes.

All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.

Isn't our PAM support optional?

PAM is the only way Virtual Desktop works now (we enter OS username and password in the login screen). Trusted is the only way embedded works now. As discussed earlier with SSO enabled both modes will have to be available for both drivers.

#73 - 07/11/2023 08:49 AM - Greg Shah

As discussed earlier with SSO enabled both modes will have to be available for both drivers.

This is my expectation as well. That seems in conflict with your comment "All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.". I'm expecting the trusted mode to be used most of the time for both web clients.

#74 - 07/11/2023 09:05 AM - Galya B

Greg Shah wrote:

That seems in conflict with your comment "All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.".

My understanding of FWD is evolving. This comment [#4129-57](#) is from 03 July 04:24 PM EET. At 07:07 PM the same day I wrote [#3931-64](#) saying:

I guess we want to make Virtual Desktop allow association of fwd users to os users without passwords with trusted spawner? Having both options for both web drivers?

And then my first report on my actual work in [#3931-69](#) is from 05 July and says:

The spawner now works with the os credentials provided in directory and verifies them. When the os user has no password, the code automatically switches to passwordless / trusted.

You're just reviewing my notes in the wrong order, at the same time I'm not supposed to delete or edit old comments.

#75 - 09/08/2023 02:23 AM - Galya B

- Status changed from New to WIP
- Assignee set to Galya B

#76 - 09/08/2023 02:23 AM - Galya B

- Status changed from WIP to Review
- % Done changed from 0 to 100

Review in progress in [#3931](#).

#77 - 10/18/2023 04:42 PM - Greg Shah

All OS users will have to have password defined for web, because the spawner does PAM authentication for the web flow.

Isn't our PAM support optional?

PAM is the only way Virtual Desktop works now (we enter OS username and password in the login screen). Trusted is the only way embedded works now. As discussed earlier with SSO enabled both modes will have to be available for both drivers.

I'd like to clarify this, because it doesn't line up with my understanding of the current spawner.

In our native spawner code for Linux, we authenticate the user's account by calling the `check_user()` function. This is done inside the `launchP2JClient()` function, just before the call to `spawn()`.

`check_user()` is implemented in 2 possible ways, using conditional preprocessing. It only uses PAM if PAM is configured at compile time. Otherwise it uses `getpwnam()` and optionally `getspnam()/crypt()` for password checking.

More importantly, it is **perfectly acceptable** for us to send NULL for the password when spawning. In both PAM mode and in non-PAM mode, this will bypass the password check and just spawn using that account. The spawner itself is installed with the suid bit set. This means that it has the rights to set the effective userid of the spawned child process. No password is needed for that feature to work.

The reason we currently prompt for the OS account password is simply because we don't want to provide a web system that allows spawning before there is any authentication step. It is not wanted in the new SSO world where we can simply spawn the FWD client in the OS account that was configured by the admin. In this new world, the application level authentication is trusted to allow the spawn.

I'm worried that the code is now written to force the storage of the password when customers will almost always want to NOT store it. Does 3931a require the password to be there? Does it work when it is not there? If not, then we definitely need to fix this before we merge.

#78 - 10/19/2023 12:37 AM - Galya B

Greg Shah wrote:

I'm worried that the code is now written to force the storage of the password when customers will almost always want to NOT store it. Does 3931a require the password to be there? Does it work when it is not there? If not, then we definitely need to fix this before we merge.

3931a doesn't require password for OS users.

#79 - 10/19/2023 12:39 AM - Galya B

You're referring to a very old comment, where I'm trying to figure out the point of the trusted security check that we finally agreed is redundant and is removed in 3931a.

#80 - 10/23/2023 11:34 AM - Galya B

- Status changed from Review to Test

#81 - 10/23/2023 11:57 AM - Galya B

3931a was merged to trunk as rev. 14783 and archived.

#82 - 02/07/2024 01:52 PM - Greg Shah

- Status changed from Test to Closed

Files

Screenshot from 2023-06-30 11-39-49.png	193 KB	06/30/2023	Galya B
---	--------	------------	---------