Runtime Infrastructure - Feature #5896

enhance legacy REST support to provide better pure Java development

12/14/2021 10:07 AM - Greg Shah

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	90%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	Yes	**version:**	
vendor_id:	GCD		

Description

History

#1 - 12/14/2021 10:10 AM - Greg Shah

This is an extension to the existing 4GL compatible REST services. Hand-written Java code that is running in the FWD server can directly use these new features to expose all remotely accessible functionality as REST services.

The following enhancements to REST support are needed:

- JSON support in BaseRecord to avoid reflection usage.
- Allow Java/POJO/DMO parameters for the services, instead of BDT.
- Enable DMO (de)serialization.
- Custom (de)serializers, etc
- Security integration for authentication at the servlet filter. See #5731-82 for details.

#3 - 02/03/2022 10:34 AM - Greg Shah

- Status changed from New to WIP

- Assignee set to Constantin Asofiei

- billable changed from No to Yes

#4 - 03/17/2022 03:03 PM - Constantin Asofiei

Eric, I have some issues with automatically serializing the DMOs. For OUTPUT, it is easy, just create the JSON.

But for INPUT, how would this work? Before the ORM refactoring (when Hibernate was involved), the DMOs were plain POJOs (so you could do a new BookImpl to create it). Now we have an entire infrastructure behind them.

So, assuming there is a dmo.p2j_test.Book INPUT parameter, how can I create a Book instance? Keep in mind that in this mode, there will be no TEMP-TABLE support, and the API's Java code will need to take this record and insert it its table.

The same issue is at the client which invokes this REST API with a Book parameter: how would this client create dmo.p2j_test.Book instances?

#5 - 03/17/2022 06:58 PM - Eric Faulhaber

If we have a factory that can create concrete instances of Book, will that meet the need? The objects it creates would be concrete instances of the dynamically assembled implementation class for the Book DMO interface. You could reference them only as Book (interface) objects and only be able to interact with them via the interface methods.

Can you show some pseudo-code or a test case that illustrates the intended use?

#6 - 03/17/2022 08:38 PM - Ovidiu Maxiniuc

If you have the DMO interface available Class<T> clazz, then all you need to do is:

```
DmoMeta dmoInfo = DmoMetadataManager.getDmoInfo(clazz);
Record record = dmoInfo.getImplementationClass().newInstance();
```

```
// eventually initialize it and set the PK;
record.initialize(session, true);
record.primaryKey(pk);
```

I created an overloaded factory method of which the most complex has the following signature:

```
public <T extends DataModelObject> T create(Class<T> clazz, boolean init, boolean defaults, Long pk)
```

The factory is constructed around a Session object it receives as parameter.

#7 - 03/18/2022 01:27 AM - Constantin Asofiei

Eric Faulhaber wrote:

> If we have a factory that can create concrete instances of Book, will that meet the need? The objects it creates would be concrete instances of the dynamically assembled implementation class for the Book DMO interface. You could reference them only as Book (interface) objects and only be able to interact with them via the interface methods.

Yes, that will be great. But, what are the constraints for it?

- will it work outside of a FWD context?
- does it require a tx to be active?

#8 - 03/18/2022 08:41 AM - Greg Shah

Is it a good idea to have the client code calculate the PK? We have very specific requirements for that value, normally it comes from our p2j_generator_sequence. Letting the caller specify anything there might have poor results later when interacting with converted code in the same table.

#9 - 03/18/2022 01:47 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Is it a good idea to have the client code calculate the PK? We have very specific requirements for that value, normally it comes from our p2j_generator_sequence. Letting the caller specify anything there might have poor results later when interacting with converted code in the same table.

Probably, no. The Session.save(BaseRecord rec, true); (sorry for syntax) will assign automatically a PK if not already set. After all, in FWD you cannot persist a record without a PK. But, if the PK was already set before the save is invoked, that specific PK will be used.

Actually I mentioned this because I encountered this usage in customer's code so I created a specific method which sets the PK. I probably should have documented it as *'optionally'* instead of *'eventually'* because the other methods do not set it.

#10 - 03/29/2022 01:52 PM - Constantin Asofiei

- % Done changed from 0 to 90

Greg, the support for the REST services implemented directly in Java is in 3821c/13714.

Please review. Also, what wiki should I use for the documentation?

Ovidiu: please take a look at p2j.persist.Record and p2j.rest.serializers.RecordSerializer.

I still need to finish the security parts.

#11 - 03/30/2022 04:06 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Ovidiu: please take a look at p2j.persist.Record and p2j.rest.serializers.RecordSerializer.

Record: no objections.
RecordSerializer:

- method initialize(): I think record.initialize() should be invoked before returning it. In case the object will not be fully read from JSON, the default values might not be the expected ones.
- method parse(): I do not know how to interpret the javadoc. What can be "null or not mutable"? This seems like a copy/paste issue.

#12 - 03/31/2022 10:18 AM - Greg Shah

Also, what wiki should I use for the documentation?

See Integrating_External_Applications

#13 - 04/04/2022 01:01 PM - Constantin Asofiei

Greg, some notes about the authentication mechanism:

- the webservice resource plugin will allow ACLs to be set for SOAP, REST, WEBHANDLER services, with a format like REST:POST:/path/to/service (the HTTP method will be included), and SOAP:POST:/path/to/endpoint:namespace/binding/operation, allowing regexps, too (/path/to/service does not include the basepath). This can set ACLs for groups or specific account, etc, and the agent side will execute the request using that account's context, so any other security plugins (like for the DMO usage or direct SQL usage) can be easily added, as the context will be properly set.
- the configuration is done at each service type (REST/SOAP/WEBHANDLER), in the directory, where an authentication node like this can appear:

```
          <node class="container" name="authentication">
             <node class="boolean" name="enabled">
               <node-attribute name="value" value="TRUE"/>
             </node>
             <!-- login and logout paths are relative to the service's basepath, or basepath/endpoint for
SOAP.
             <node class="string" name="login_path">
               <node-attribute name="value" value="/fwdlogin"/> <!-- optional, if not set it means each AP
I call will need to authenticate and timeout needs to be 0 -->
             </node>
             <node class="string" name="logout_path">
               <node-attribute name="value" value="/fwdlogout"/> <!-- optional, defaults to /fwdlogout if
login_path is set -->
             </node>
             <node class="string" name="type">
               <node-attribute name="value" value="basic|oauth1"/>
             </node>
             <node class="integer" name="timeout">
               <node-attribute name="value" value="3600"/> <!-- if '0' is used, each API call will need to
 authenticate -->
             </node>
          </node>
```

  where:
  - enabled is a flag to enable or disable authentication for this service type
  - login_path and logout_path are optional and relative to the basepath for REST/WEBHANDLER or endpoint for SOAP; if login_path is not set, then each API call will have to provide the authentication details, and the runtime will simulate a login, API invoke, logout. The context will not survive this call.
  - type is the authentication type - only basic is supported at this time. Oauth1 is more complex, and if there is a need, it can be added easily.
  - timeout is used when the login_path is specified: after login, the context will be created and saved, and a token will be returned which will need to be set to any other API calls; all API calls using that token will use the same FWD context. This must be non-zero, as when the timeout elapses, the FWD context/session will be destroyed.
- For basic auth mode, the approach will be to set the Authorization header with the base64-encoded user:token, where token is an UUID set at the account (webServiceToken).
- session usage tracking (and limit) is done only for user accounts, and I'd like to allow this. So only FWD user accounts can have this webServiceToken. But this gives another complexity, as the user account has also a password - I think an user account with webServiceToken should not be allowed interactive/password login.
- concurrency issues: FWD contexts do not support this. So I need to prohibit an API call (or queue them?) if there is already an API call for that token... I'm not sure yet how to solve this, but this is where the FWD session usage tracking comes into play.

#14 - 04/05/2022 12:10 PM - Constantin Asofiei

Greg, something which should have been obvious for me from the start: the agent will switch context and execute the target API using whatever FWD account matches the authentication set at the request, but this will not allow any client-side features, like memptr, streams, etc, as there is no matching FWD client process - that belongs to the agent. Should I spawn a FWD client for this context, too?

Otherwise, the authentication and authorization changes work, and the agent executes the target in that context.

#15 - 04/05/2022 01:57 PM - Constantin Asofiei

The changes to add authentication and authorization for web services are in 3821c/13740.

I'll finish the documentation tomorrow.

#16 - 04/06/2022 01:55 PM - Constantin Asofiei

Greg, please review Java REST API and Securing the Web Services, and let me know what else should I add.

#17 - 04/07/2022 10:48 AM - Greg Shah

I have reviewed the documentation. Here are some questions/comments about the implementation:

Overall, this is very impressive! It is a powerful and elegant approach to expose an API for a FWD server.

1. In the name map merge processing, re-using the class-mapping entry leads to the confusing case where there is a bogus pname attribute. Considering there is no .p associated with this pure Jave REST service, including this will only cause questions and errors. I don't think it is helpful. Instead, let's implement a simpler approach which uses a different element like java-rest-service or rest-service instead of class-mapping. You could get rid of both the pname and with-services and just code the jname.

2. I notice that the example JavaRestTest and SimpleRestTest is an unqualified class name.

- What is the search process for these names?
- Do we support a fully qualified name here?

3. I assume it is OK to specify both input and output for the annotations, which would be equivalent to input-output in 4GL code. If that is correct, please make it explicit in the documentation. If not, please still make it explicit.

4. The JavaTypeSerializer.parse() might be better named fromJson().

5. I think it would be helpful to show some Java examples:

- simple scalar arguments
  - input args
  - output args
  - input-output args (if possible)
- array, set and map arguments
  - input args
  - output args
  - input-output args (if possible)
- DMO usage

Each of these probably should also so the client-side call and the JSON response.

6. The following lists would be easier to read/reference as tables:

- Java class LegacyService annotation settings (columns: name, data type, required, description)
- Java method LegacyService annotation settings (columns: name, data type, required, description)
- input arguments (columns: name, data type, description)
- output arguments (columns: name, data type, description)
- parameter data types (columns: data type, input, output, serialization details)

7. There is a comment "authentication and authorization is not mandatory". Does this mean that by default everything is open (unsecured)? That is something we want to avoid. Also, in that case what context is used to execute the service?

8. Should we consider using process account certificates as a means of authentication? Under the appserver approach using DAP, each remote API user would be a process account and would be identified as part of the session setup using certificates. Instead of adding the webServiceToken to the account, what is a scenario where we leverage the same certificates approach as we already use? I'm not opposed to keeping a simple webServiceToken approach as an option. I just think there are benefits to the certificates approach.

#18 - 04/07/2022 10:55 AM - Greg Shah

> concurrency issues: FWD contexts do not support this. So I need to prohibit an API call (or queue them?) if there is already an API call for that token... I'm not sure yet how to solve this, but this is where the FWD session usage tracking comes into play.

I expect that we do need to support concurrency. In fact it is very likely that will be the common case, where some application has multiple threads accessing an FWD hosted in a FWD server. They will not want (or need) to configure different accounts for each thread.

I would expect that each API request would execute in an agent from the pool and that the user context would be assigned to the agent temporarily, the way we do with dispatching appserver requests. Are we doing this differently now for REST/SOAP/WebHandler?

> Greg, something which should have been obvious for me from the start: the agent will switch context and execute the target API using whatever FWD account matches the authentication set at the request, but this will not allow any client-side features, like memptr, streams, etc, as there is no matching FWD client process - that belongs to the agent.

I guess I don't understand the exact approach yet, because I assumed we would be dispatching into the agent pool and each agent already has a client.

On the other hand, I do not object to exluding client usage. I think that would be a very good thing for newly written services to be server-only. This may not be feasible for services that call legacy code.

> Should I spawn a FWD client for this context, too?

Help me understand the current model, then we will discuss this issue.

#19 - 04/07/2022 11:11 AM - Constantin Asofiei

Greg Shah wrote:

> Help me understand the current model, then we will discuss this issue.

The authentication and authorization implementation does this:

- the request has a FWD username and a token. FWD looks for a user with that name and checks if the tokens match - if this is correct, then the authentication passes.
- once the authentication passes, FWD will create a context for that user. Authorization is done using the webservice ACLs, to check if the user is allowed to access that web service.
- if the user is authorized, then the target API for that web service is still dispatched to an agent, but that agent doesn't execute the request in its own context: it switches the context to the user authenticated for this web request, and executes it. This is the only way to add other security mechanisms, like at the ORM layer, to see if it is authorized to i.e. read a certain table. At the time the authorization for the web service happens, there is no way to know what the actual implementation does, to perform further authorization, using i.e. ORM layer security. This is the sole reason why the agent switches context.

  7. There is a comment "authentication and authorization is not mandatory". Does this mean that by default everything is open (unsecured)? That is something we want to avoid.

Yes, by default everything is open (unsecured) and is the application's responsibility to secure the APIs.

  Also, in that case what context is used to execute the service?

The agent's context.

  8. Should we consider using process account certificates as a means of authentication? Under the appserver approach using DAP, each remote API user would be a process account and would be identified as part of the session setup using certificates. Instead of adding the webServiceToken to the account, what is a scenario where we leverage the same certificates approach as we already use? I'm not opposed to keeping a simple webServiceToken approach as an option. I just think there are benefits to the certificates approach.

I thought about this, but how will the certificate be sent? Via some header? This just means adding a new way of authenticating beside 'basic', it will not be difficult to plug it in. The 'Basic' authentication is something simple which can be setup using SoapUI or Postman, using the certificate is more complex.

#20 - 04/07/2022 11:17 AM - Constantin Asofiei

> 7. There is a comment "authentication and authorization is not mandatory". Does this mean that by default everything is open (unsecured)? That is something we want to avoid.

Another note here: we can't force the FWD authentication and authorization to be mandatory, we have current applications which have its own security builtin, at the request (like Oauth1). If we force this authentication, then these 3rd party applications which perform the web service request will need to be changed, to include the FWD authentication/authorization, beside the application's.

#21 - 04/07/2022 11:42 AM - Greg Shah

The authentication and authorization implementation does this:

- the request has a FWD username and a token. FWD looks for a user with that name and checks if the tokens match - if this is correct, then the authentication passes.
- once the authentication passes, FWD will create a context for that user. Authorization is done using the webservice ACLs, to check if the user is allowed to access that web service.
- if the user is authorized, then the target API for that web service is still dispatched to an agent, but that agent doesn't execute the request in its own context: it switches the context to the user authenticated for this web request, and executes it. This is the only way to add other security mechanisms, like at the ORM layer, to see if it is authorized to i.e. read a certain table. At the time the authorization for the web service happens, there is no way to know what the actual implementation does, to perform further authorization, using i.e. ORM layer security. This is the sole reason why the agent switches context.

So: if the calling application has multiple threads, each thread will execute the login with the same user account and webServiceToken but will get a different "access" token which will be set in the FwdSessionId header of the request.

That means that there are separate contexts created for each thread and the subequent agent processing is safe. There is no concurrency issue. If I understand correctly, that is all good.

Your point about concurrency was just that each authenticated session should only be used for one request at a time?

> 7. There is a comment "authentication and authorization is not mandatory". Does this mean that by default everything is open (unsecured)? That is something we want to avoid.

Yes, by default everything is open (unsecured) and is the application's responsibility to secure the APIs.

I do think this is an issue, we should not allow any access by default.

> Another note here: we can't force the FWD authentication and authorization to be mandatory, we have current applications which have its own security builtin, at the request (like Oauth1). If we force this authentication, then these 3rd party applications which perform the web service request will need to be changed, to include the FWD authentication/authorization, beside the application's.

I agree that we cannot expect calling applications to be rewritten to our authentication model. I'm not suggesting this. I'm saying that we should require that the FWD server is explicitly configured to allow access, even if that configuration is just telling FWD: "open access to all". If we need to make changes to allow this configuration to be done with little effort, I am open to that.

#22 - 04/07/2022 01:37 PM - Greg Shah

Code Review Task Branch 3821c Revisions 13714, 13740

It is very good. I especially like the extensibility of the serializers and the clean approach for the web security plugins.

1. The reasoning behind the use of ThreadLocal in JsonRequestsParser, ResponseArguments is not obvious. Can you explain the idea?

2. Did you use regular user accounts because of the limitations of process accounts? In most ways, process accounts are a better match. The certificates part is the only issue. I guess this is why there is now a kind of "sub-type" of user, which is a web session. I'm still getting my head around it. I'm not entirely happy with the need for it and how it changes the SecurityManager.

3. LegacyArgumentsParser constructor and parseArgument() needs javadoc.

4. LegacyArgumentsSerializer needs javadoc.

#23 - 04/07/2022 01:57 PM - Constantin Asofiei

Greg Shah wrote:

> 1. The reasoning behind the use of ThreadLocal in JsonRequestsParser, ResponseArguments is not obvious. Can you explain the idea?

An instance of ResponseArguments or JsonRequestsParser can be used by multiple requests - this lets me re-use that instance and not create one for each request. For JsonRequestsParser, is more a 'better safe than sorry', I'm not fully sure that the ObjectMapper is completely thread-safe. For ResponseArguments, the serializer field must be thread-local.

> 2. Did you use regular user accounts because of the limitations of process accounts? In most ways, process accounts are a better match. The certificates part is the only issue. I guess this is why there is now a kind of "sub-type" of user, which is a web session. I'm still getting my head around it. I'm not entirely happy with the need for it and how it changes the SecurityManager.

The main reason was the requirement for 'basic authentication' to use a token, which IMO is better at the user account. I don't have a complete picture what configuration would be to add OAuth1, but we may want to consider to add a dedicated 'web user' account.

#24 - 04/07/2022 02:04 PM - Constantin Asofiei

Greg Shah wrote:

> So: if the calling application has multiple threads, each thread will execute the login with the same user account and webServiceToken but will get a different "access" token which will be set in the FwdSessionId header of the request.

Yes.

> That means that there are separate contexts created for each thread and the subequent agent processing is safe. There is no concurrency issue. If I understand correctly, that is all good.

Yes, there is a context created for each access token.

> Your point about concurrency was just that each authenticated session should only be used for one request at a time?

The same access token can be used by parallel requests, that's why I added the protection in Agent.activeWebServiceTokens. Multiple requests can be started at the same time, but they will be queued, waiting for the access token to be released.

> I agree that we cannot expect calling applications to be rewritten to our authentication model. I'm not suggesting this. I'm saying that we should require that the FWD server is explicitly configured to allow access, even if that configuration is just telling FWD: "open access to all". If we need to make changes to allow this configuration to be done with little effort, I am open to that.

Please elaborate what 'open access to all' means in a configuration POV.

- Do you want the WebServiceResource security plugin to be loaded by default in FWD (this can be done with no side-effect)?
- a default at the authentication config for the web service type, where security is enabled (and timeout, login path, etc are set to some defaults), and you would have to disable it explicitly?

What I think is missing is some granularity on what web service paths should be authenticated and which should not. This would better match your 'open access to all' idea, to have some security plugin which by default allows all paths without authentication, and have explicit paths where authentication is enabled. Another reason for this would be the 'no FWD client exists' at the created context for the web request, as currently I can't distinguish between legacy REST services (which would require client-side features) and Java REST services (which can be built without client-side dependencies).

#25 - 04/13/2022 11:27 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> RecordSerializer:
>
> - method initialize(): I think record.initialize() should be invoked before returning it. In case the object will not be fully read from JSON, the default values might not be the expected ones.

Do you have an example where you can work with FWD Record instances, outside of their buffer? Also, at the time these instances are created, we are outside of a transaction; so the ORM session can't and shouldn't track these. If you want to save these records, then you need to either create a record buffer and copy the changes from this Record to the buffer's record, or attach the record to the FWD session and persist it after that.

I agree that the defaults should be set, I'll extract that code from Record.initialize(Session, boolean setDefaults) into a initialize(setDefaults).

> - method parse(): I do not know how to interpret the javadoc. What can be "null or not mutable"? This seems like a copy/paste issue.

I'm improving this.

#26 - 04/20/2022 03:08 PM - Constantin Asofiei

3821c/13800 contains improvements from the #5896-17 review.

The docs are updated and examples added in Java REST API .

I haven't improved the authentication/authorization, I don't have a clear picture yet what and how should be changed. At least, maybe we need another security plugin to specify which paths require authentication and which not.

#27 - 04/21/2022 08:53 AM - Greg Shah

Code Review Task Branch 3821c Revision 13800

The changes are good.

#28 - 04/21/2022 09:02 AM - Greg Shah

> I agree that we cannot expect calling applications to be rewritten to our authentication model. I'm not suggesting this. I'm saying that we should require that the FWD server is explicitly configured to allow access, even if that configuration is just telling FWD: "open access to

all". If we need to make changes to allow this configuration to be done with little effort, I am open to that.

Please elaborate what 'open access to all' means in a configuration POV.

- Do you want the WebServiceResource security plugin to be loaded by default in FWD (this can be done with no side-effect)?

Yes. We probably should all the ones that are built into FWD. I see no reason these should be optional. It will reduce the configuration requirements.

- a default at the authentication config for the web service type, where security is enabled (and timeout, login path, etc are set to some defaults), and you would have to disable it explicitly?

Yes.

What I think is missing is some granularity on what web service paths should be authenticated and which should not. This would better match your 'open access to all' idea,

I don't want 'open access to all'. This is exactly my concern. Progress may be open by default but we must not be. I want the opposite, 'no access by default, only access that is configured is allowed'.

to have some security plugin which by default allows all paths without authentication, and have explicit paths where authentication is enabled. Another reason for this would be the 'no FWD client exists' at the created context for the web request, as currently I can't distinguish between legacy REST services (which would require client-side features) and Java REST services (which can be built without client-side dependencies).

Do we need to enhance the annotations for the Java REST services to allow this to be detectable?

I haven't improved the authentication/authorization, I don't have a clear picture yet what and how should be changed. At least, maybe we need another security plugin to specify which paths require authentication and which not.

If my answers above did not help, please ask more questions.

#29 - 04/21/2022 09:18 AM - Constantin Asofiei

Greg Shah wrote:

> We probably should all the ones that are built into FWD. I see no reason these should be optional. It will reduce the configuration requirements.

I'll need to go through all the projects and see what/how is enabled, and which are safe to be built in. Some may require mandatory ACLs to enable access for i.e. spawner.

> > to have some security plugin which by default allows all paths without authentication, and have explicit paths where authentication is enabled. Another reason for this would be the 'no FWD client exists' at the created context for the web request, as currently I can't distinguish between legacy REST services (which would require client-side features) and Java REST services (which can be built without client-side dependencies).
>
> Do we need to enhance the annotations for the Java REST services to allow this to be detectable?

I don't think this is needed at the annotation, but a security plugin with ACLs configured for certain paths. The i.e. unsecured or secured APIs can be grouped under certain path(s). So you could have something like 'no authorization for all paths' and after that ACLs with more granular 'authorization for these paths'. Or viceversa, 'authorization for all paths' and granular ACLs with 'no authorization for these paths'.

> If my answers above did not help, please ask more questions.

I think is clearer now, the point is to 'force' the user to 'open access to all' for certain APIs, so these are explicitly visible in the directory.

#30 - 04/21/2022 09:27 AM - Greg Shah

> the point is to 'force' the user to 'open access to all' for certain APIs, so these are explicitly visible in the directory.

Correct.

> So you could have something like 'no authorization for all paths' and after that ACLs with more granular 'authorization for these paths'.

Yes, this is the way. There is no authorization except what is explicitly added to ACLs.

#31 - 04/21/2022 11:13 AM - Greg Shah

This approach requires an explicit intervention to open everything. But that intervention is not hard. It is just a single ACL with * as the resource and the subject as all_others.

#32 - 04/21/2022 11:14 AM - Constantin Asofiei

Greg Shah wrote:

> This approach requires an explicit intervention to open everything. But that intervention is not hard. It is just a single ACL with * as the resource and the subject as all_others.

Exactly, or just disable the authentication at the rest/soap/webhandler nodes in the directory.

#33 - 04/21/2022 11:18 AM - Greg Shah

> or just disable the authentication at the rest/soap/webhandler nodes in the directory.

What is the advantage of keeping this? I think it can only confuse and maybe cause errors. Normally, we don't allow resources to be defined as "not secured".

#34 - 04/21/2022 11:38 AM - Constantin Asofiei

Greg Shah wrote:

> > or just disable the authentication at the rest/soap/webhandler nodes in the directory.

> What is the advantage of keeping this? I think it can only confuse and maybe cause errors. Normally, we don't allow resources to be defined as "not secured".

I'll remove it, too much configuration will be confusing.

#35 - 04/22/2022 06:38 AM - Constantin Asofiei

If I force defaults for rest/authentication/login_path and rest/authentication/logout_path, to /fwdlogin and /fwdlogout, then I don't see how I can allow authentication per each request (this will force to use the login/logout paths to get the login token).

#36 - 04/22/2022 12:05 PM - Greg Shah

Constantin Asofiei wrote:

> If I force defaults for rest/authentication/login_path and rest/authentication/logout_path, to /fwdlogin and /fwdlogout, then I don't see how I can allow authentication per each request (this will force to use the login/logout paths to get the login token).

What do you mean by "force defaults for"?

#37 - 04/22/2022 12:08 PM - Constantin Asofiei

Greg Shah wrote:

> Constantin Asofiei wrote:
>
> > If I force defaults for rest/authentication/login_path and rest/authentication/logout_path, to /fwdlogin and /fwdlogout, then I don't see how I can allow authentication per each request (this will force to use the login/logout paths to get the login token).
>
> What do you mean by "force defaults for"?

Currently rest/authentication/login_path is optional - if is missing, then each request must contain the auth details.

If I use defaults for login_path, this mode where authentication is done at each call will no longer work.

As a reminder, currently there are two ways for the authentication to work:

- via /fwdlogin where a token is returned, and this is used with all other requests
- if login_path is not set, the authentication details must be set for each request

#38 - 04/22/2022 12:08 PM - Constantin Asofiei

The structure of an ACL to check if a path must be authenticated is:

```
        <node class="container" name="webauth">
          <node class="container" name="000100">
            <node class="resource" name="resource-instance">
              <node-attribute name="reference" value=".*"/>
              <node-attribute name="reftype" value="FALSE"/>
            </node>
            <node class="webAuthRights" name="rights">
              <node-attribute name="allow" value="FALSE"/>
            </node>
            <node class="strings" name="subjects">
              <node-attribute name="values" value="all_others"/>
            </node>
          </node>
```

Here, the 'subjects' has no meaning - as the check is not done against a subject, is done against the path. And this is done pre-authentication, so there is no subject or FWD context anyway. I don't see how this can be enforced in the FWD code, only way I see is to document 'use all_others at subjects'. Also, the path here will not be relative to the service basepath or address, will be the full path.

Another problem here is this check should be done in the Jetty request thread, and there will be no FWD context.

#39 - 04/22/2022 12:31 PM - Greg Shah

> If I use defaults for login_path, this mode where authentication is done at each call will no longer work.

What is an example of a default? I don't understand how this relates to the security model. It is OK to have 2 approaches to authentication.

> Here, the 'subjects' has no meaning - as the check is not done against a subject, is done against the path.

This doesn't seem right. It is only the combination of resource + subject where can tell if the subject is allowed access to the resource. You must know both.

> And this is done pre-authentication, so there is no subject or FWD context anyway. I don't see how this can be enforced in the FWD code, only way I see is to document 'use all_others at subjects'.

I think login would be special since it is required to get to a context. It should always be exposed without the need for an ACL. But all other resources must be protected by context.

> Also, the path here will not be relative to the service basepath or address, will be the full path.

Please show examples.

> Another problem here is this check should be done in the Jetty request thread, and there will be no FWD context.

I thought the idea was that a login happens (which creates a context) or the token is passed which identifies an existing context. Either way, once the context is available it must be used for the check.

#40 - 04/22/2022 12:42 PM - Constantin Asofiei

Greg, this is a 'chicken and the egg' problem. Lets say an application secures only the /rest/private/ paths and leaves /rest/public/ for public access.

These paths should be configured via ACLs in the directory, to know which has enabled authentication and which has not. Now, when I need to check that /rest/private/ requires authentication, there is no context set (yet). It doesn't matter if the token exists at the header.

Note that there are two steps to allow access to a secured path:

1. authentication, where the subject's context is created, if the authentication was possible
2. authorization, where FWD checks if that subject is allowed access to a secured API.

When a path is not secured, no authentication or authorization is required, thus there is no dedicated context or subject for that request.

#41 - 04/22/2022 01:27 PM - Greg Shah

OK, at a minimum, it seems like we need a list of those paths which are unsecured. These are publicly visible and require no authentication for access.

For secured APIs:

- Should there be a differentiation between publicly visible and only visible to already authenticated contexts?
- This is where we need to mark them based on whether there is a login_path or not.