


Crash Course on Security

A critical component of any system is its security model. It's important to identify the different types of users that you have, and what each user type should be granted access to see and do inside the system.

In Onit, the *default* approach to security is always one of least access possible. That is, unless you configure otherwise, the only access that any given user will have is to the Records to which he/she is a Participant of. In most cases though, you'll want to create a more robust and customized security model for your Onit implementation.

To do so, Onit offers multiple, overlapping layers of security. In this tutorial, we'll cover each possible layer:

- [Participants](#)
- [Admin Roles](#)
- [User Groups](#)
- [Private Access User Group](#)
- [The "Restrict Access" checkbox](#)
- [Security Chains](#)
- [Liquid Conditions](#)

 **Note:** As is true for any robust platform, Onit security is an especially advanced topic. This tutorial is aimed at technical administrators who have both set up security in other similar systems and are also generally familiar with standard Onit concepts (like creating and managing apps and users).

Overlapping Security Layers

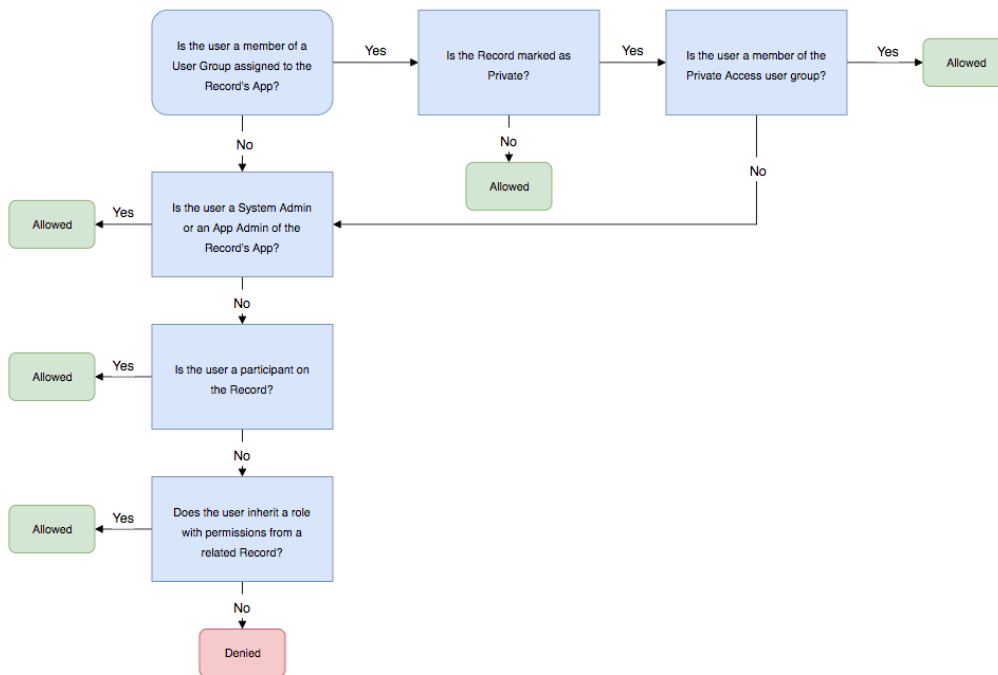
Before we get too deep into the details, it's critical to understand that Onit's security layers are designed to overlap with one another. This means that even though a particular layer doesn't grant the ability to do something, another layer may.

For example, let's say that you grant a certain security layer to a user named Bob, which grants him the ability to do **X** but not **Y**. Does this mean that Bob definitely cannot do **Y**? Not necessarily as Bob may have been granted **Y** via an entirely different security layer.

Bottom-line: Always take a holistic view of all possible security layers at play.

If you're interested in what security layers trump others, you can refer to the diagram below:

Does a User Have Access to a Record?



Security Layer: Participants

This is the most basic layer of security. The easiest way to grant someone access to a transaction is simply to ensure that they are one of its Participants. The permissions that they will have on the transaction depend on their Role. For example, if their Role is limited to read-only access, Participants added via this Role will have read-only access.

Important: Participants are a fundamental Onit concept, which you should be aware of before continuing with this tutorial. If this is a new concept to you, check out [Building Your First App](#). To learn more about adding Participants to transactions, see [Adding a Participant from a Field Value](#) and [Adding Participants from a Decision Table](#).


Security Layer: Admin Roles

This layer of security is primarily aimed at administrators, not regular users.

Using predefined Admin Roles, you can grant users various degrees of administrative privileges which apply to an entire corporation or app.

There are five different pre-defined Admin Roles available, each of which is described below (each predefined Admin Role appears in blue in the following table).

	System Administrator	App Creator	List Administrator	API User	App Administrator
Applies to the entire corporation or to a specific app?	Entire Corporation				Specific App Note: A user can be assigned this Admin Role for multiple apps.
Can manage corporation-level administrative settings?	Can manage all possible corporation-level settings.	Cannot manage any corporation-level settings or view the Administration page.	Can only manage a corporation's Lists (no other corporation-level settings).	Cannot manage any corporation-level settings or view the Administration page.	
Can view, modify, and delete all transactions?	Yes	No. Does not grant permission to view, modify, or delete any transactions.		Yes, but only via authenticated API calls. (Does not grant permissions within the Onit user interface.)	Yes, for the specific app in question.
Can perform app-level and suite-level configuration on <u>all</u> apps? Note: This includes creating and deleting all apps/suites.	Yes		No. Does not grant permission to open any app's Wizard or Advanced Designer page.		

 To learn how to assign a user an Admin Role, see the **Providing Users with Administrative Roles** section of the [Adding and Managing Users](#) tutorial.



Note: A user can be assigned *multiple* Admin Roles simultaneously.



Important: The Admin Roles discussed here are entirely different than the Roles that you create within a **Wizard**. Other than both having the word “role” in their name, these two entities have nothing in common.

Security Layer: User Groups

The primary purpose of this security layer is to grant users Roles Based Access Control (RBAC) to transactions, even if they aren't transaction Participants.

For example, let's say that your **Contract** App automatically (and conditionally) assigns a few Participants to every transaction, such as a Requester, a Legal Approver, and a Finance Approver. Let's also say, however, that there are a static group of administrators that need access to every single transaction.

In this situation, you could configure Onit to auto-assign each administrator to every transaction (using an Action), but that would be cumbersome to set up and manage (especially when admins were hired, fired, and/or transferred to different jobs/departments). Instead, it would be much easier to simply tell Onit, “I'll create a group named **Admins**. Make sure that anyone in that group can see all transactions in my **Contracts** app -- even if they aren't Participants.”

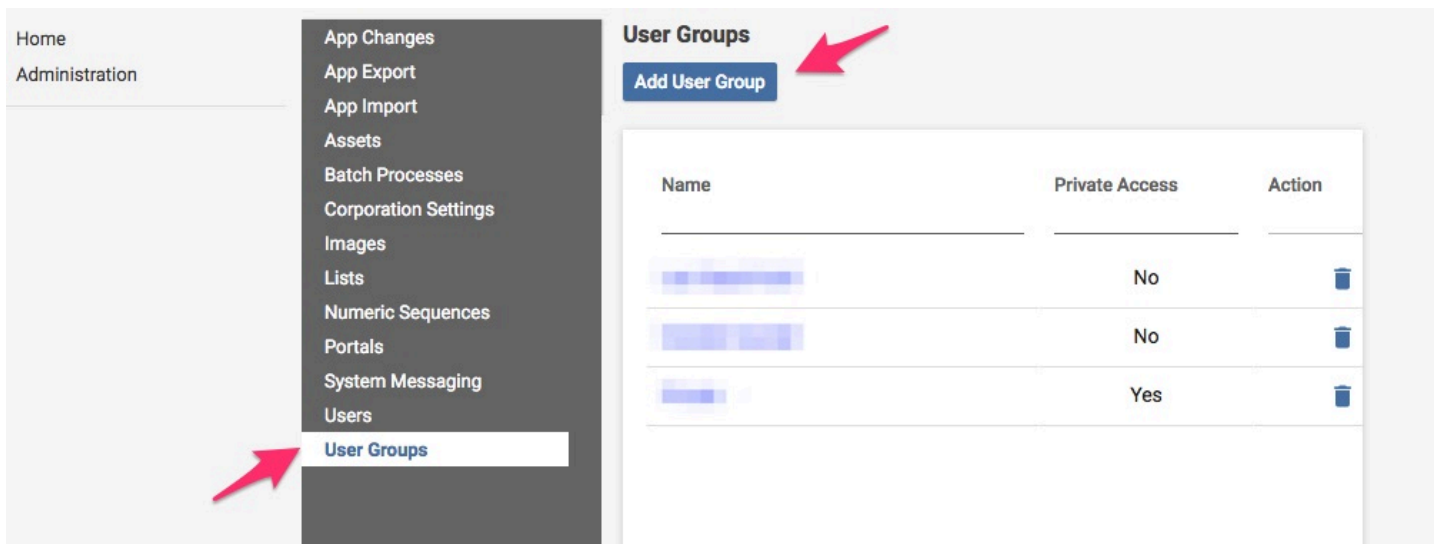
To set this up, you'll need two things:

- A User Group that you created (like **Admins** from our example above). This is created at the corporation-level, and is then available to all apps.
- A Role that you created. This is created at the app-level, inside of an app's **Wizard**.

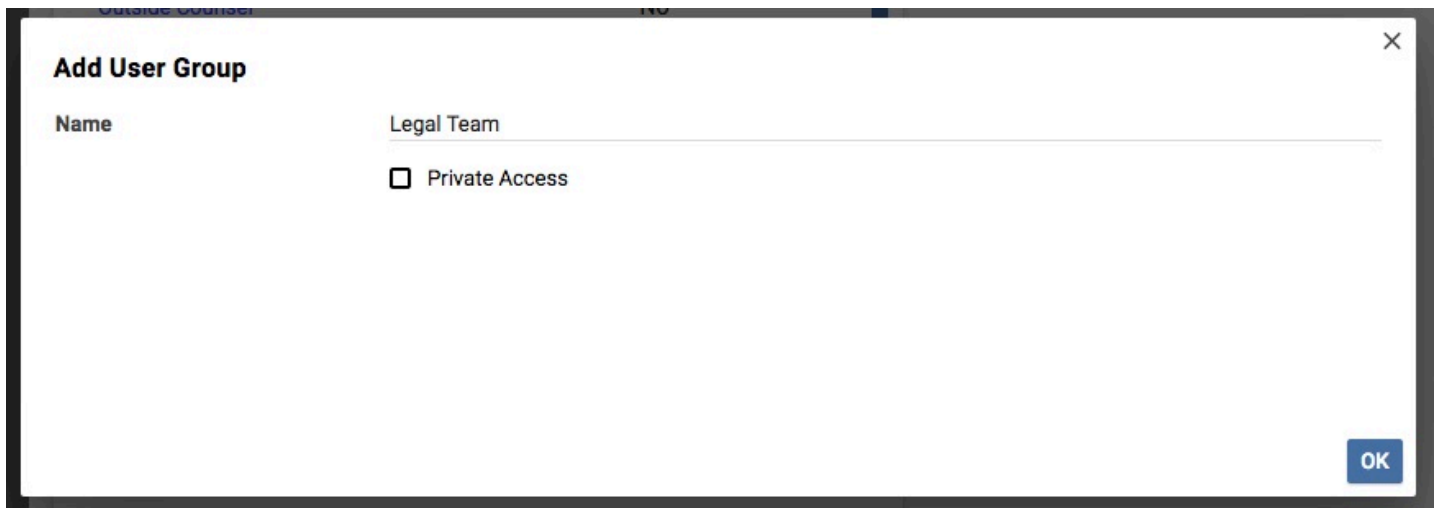
Once these are set up, browse to an app's **Advanced Designer** page and assign your User Group to your app. When you do so, you'll assign it one of the app's Roles, which controls what users in that User Group can do on any given transaction (e.g., if the Role is limited to read-only access, that permission will filter down to the User Group).

1. Create a User Group

Browse to your environment's **Administration** page and select **Add User Group** from the left-hand pane.



Provide it with a **Name** and leave **Private Access** unselected.



Select **OK**.

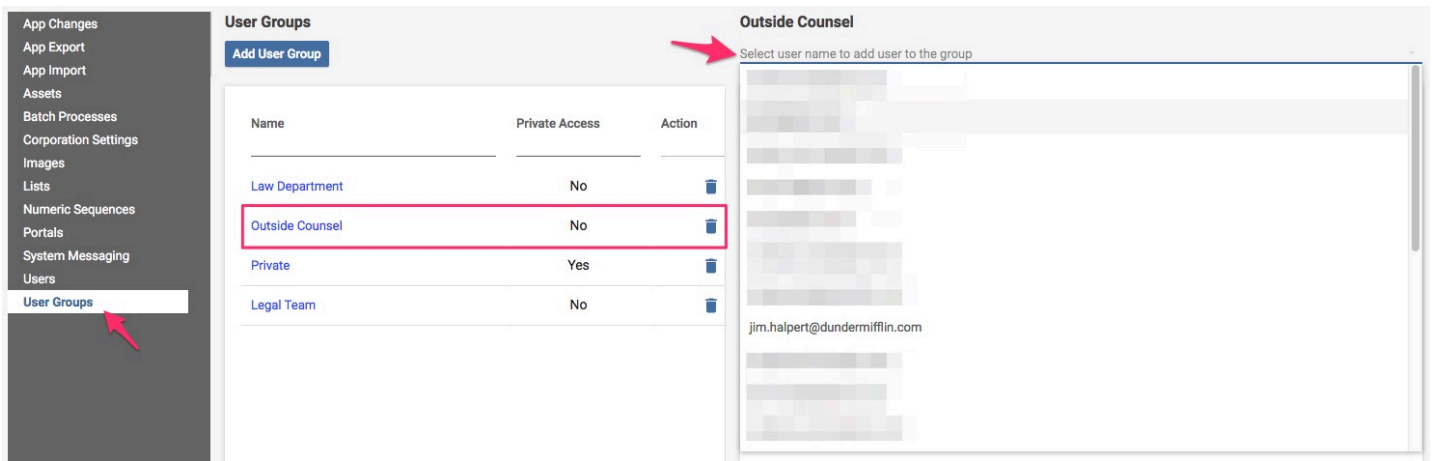
2. Add Users to the User Group

Still on the environment's **Administration** page, select **User Groups** from the left-hand pane.

Select the **User Group** that you want to add users to.

Select **Users**.

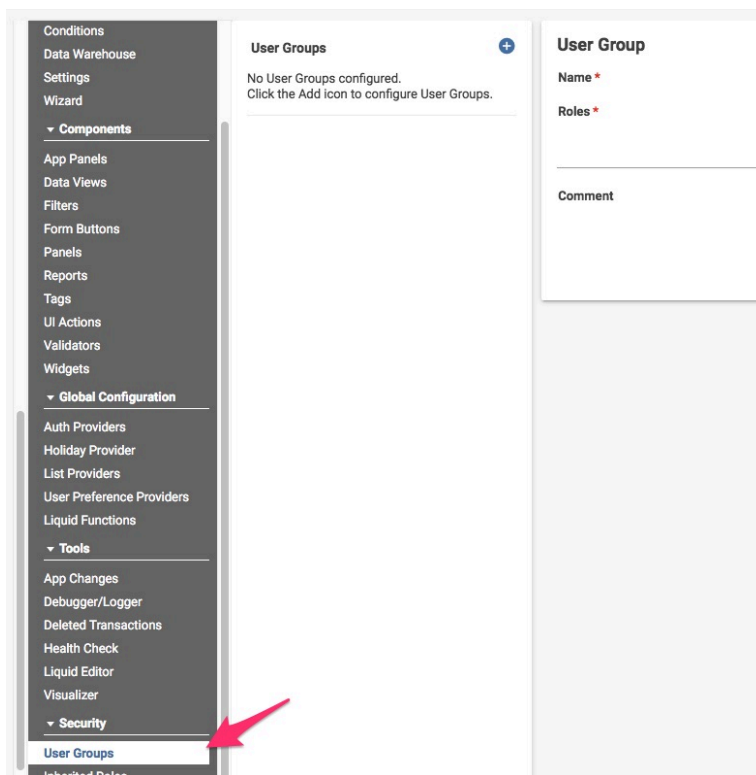
Using the dropdown at the top of the page, add users to the group.



3. Browse to an App and Assign the Group to it

Lastly, you'll need to assign the group to the App whose Records you want the users in the User Group to have access to. Jump over to this App's **Advanced Designer** page.

From the left-hand pane, select the **User Groups** node.



From the **Name** dropdown, select the group that you just created.

Finally, select one or more **Roles**.

User Group

Name *

Legal Team

Roles *

Legal Approver

Comment

Provide a brief description of the change you made (optional)

Delete

Save

Click **Ok** to save your changes.

Note: If you add one role that is limited to read-only access and another role that has read/write access, the group will have read/write access.

Note: Users in a group will only see the Tabs that the Role associated with the User Group has been assigned to.

That's all there is to it. The users in your User Group should now have the same permissions to transactions that the Role you assigned to it does.

Security Layer: Private Access User Group

In the previous section, we covered User Groups in general. In this section, we'll discuss a special type of User Group, which is considered a separate layer of security.

In all but one situation, User Groups apply to *all* Records in an App. In other words, by assigning a User Group to your App, you're granting that group access to every Record. The exception to this rule, however, is a User Group that has been marked "private access," which allows you to control security on a Record-by-Record basis.

Let's consider an example where this security layer might be helpful. Let's say that you have a User Group named **Law Department**, which you've added to your App. As a result, all members of this group have access to every single Record. After setting this up, you realize that this approach might be too generous, because there are a small number of Records that are especially sensitive in nature. In this situation, you could use the Private Access security layer to limit access to a very small number of users, whom you can trust with sensitive data.

To set up this security layer, you need to do two things:

- Create a checkbox named **private** in your App.
- Create a User Group in your corporation named **Private** that has been marked "private access".

Using this set up, when a Record's **Private** checkbox is checked by a user, the Record in question will only be visible to users that meet at least one of the following conditions:

- Are members of the User Group that has been marked "private access"
- Are a Participant on the Record

Note: There are multiple ways our security model allows a user to access a Record. If any of these ways allow a user to access a Record, that user has access to a Record. If a user has been assigned an Admin Role of either **System Administrator** or **App Administrator** (for the App in question), they will be able to see, modify, and delete all Records, even if they are not in the **Private** User Group. Likewise, if you've added an inherited role to your App (explained in the next section of this tutorial), that will allow users who have access to Record in the inherited App to access related Records in your App, regardless of whether they're marked private. Always be sure to test your security model exhaustively, using different user personas.

Let's go through the step-by-step process of configuring the Private Access security layer.

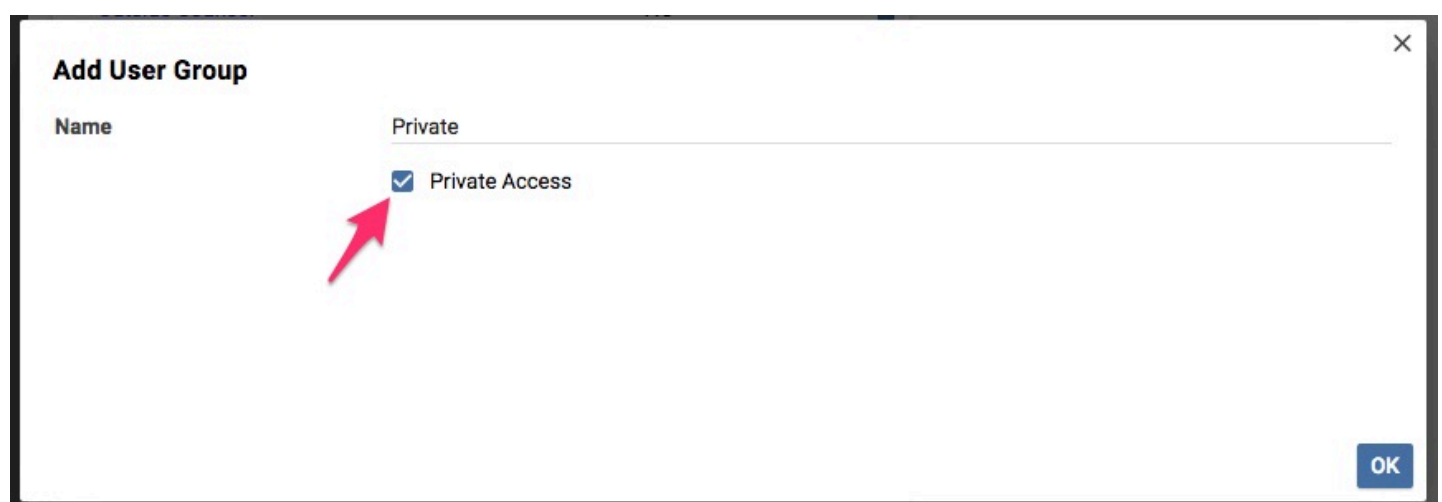
1. Create and Populate the "Private" User Group

Browse to your environment's **Administration** page and select **Add User Group** from the left-hand pane.

Provide it with a **Name** of **Private**.

Note: The name of this User Group can technically be anything you like. That said, it is a common Onit convention to name this User Group **Private**.

Additionally, select **Private Access**. (Don't forget this step, as it's critical.)



The screenshot shows a dialog box titled "Add User Group". It has a close button (X) in the top right corner. The "Name" field contains the text "Private". Below the name field, there is a checkbox labeled "Private Access" which is checked. A red arrow points to this checkbox. In the bottom right corner, there is a blue button labeled "OK".

Select **OK**.

Proceed to add the users to this User Group who you want to have access to transactions marked as private.

2. Create a Private Checkbox

Browse to an app, open its **Wizard**, and proceed to the **Fields** screen.

Add a new Field with a **Type** of **Checkbox**.

Most importantly, give this Field a **Name** of **private**. (Be sure to use this *exact* name.)

The screenshot shows the 'Fields' screen in the Wizard. The 'Fields' tab is selected. A table lists fields, with 'private' highlighted. The 'Properties' panel on the right shows the configuration for the 'private' field.

L	V	Name	Display Name	Data Type
2	2	private	Private	Checkbox

Properties

Display In: Launch and View

Launch Page Tab: Create Request

View Page Tab: General

Tags

Name: private

Label: Private

☐ Hide Label

Label Width

☐ Include Tooltip

Tooltip Text

Data Type: Checkbox

Right Label

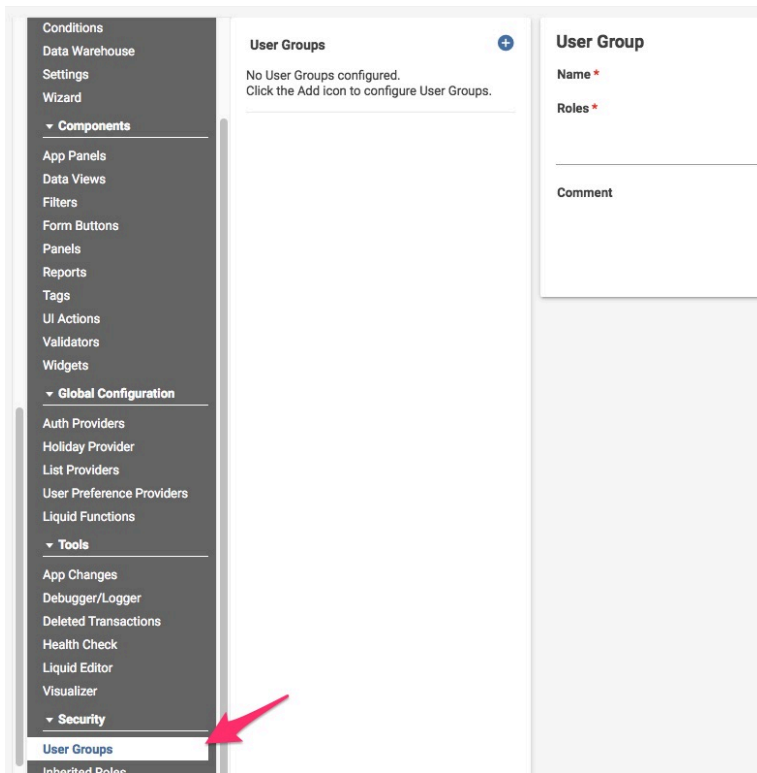
Initial Value

UI Action

Update your app to save your changes.

3. Assign "Private" to Your App

In your App's **Advanced Designer** page, from the left-hand pane select the **User Groups** node.



From the **Name** dropdown, select the “private access” User Group that you just created above.

Select one or more Roles and click **Ok** to save your changes.

User Group

Name *

Private

Roles *

FYI

×

Comment

Provide a brief description of the change you made (optional)

Delete

Save

You're done!

If you check any transaction's **Private** checkbox, only its participants and the users in your **Private** User Group will have access to it.

The "Restrict Access" checkbox:

💡 You might wonder, now that I have set up my User Groups how do I prevent other users from accessing and creating Records in Apps where they shouldn't be? The answer is the **Restrict Access** checkbox.

The Restrict Access checkbox is the official platform way to prevent users from creating records in an App unless the user belongs to an associated User Group. Set up is very simple; Navigate to your App's Advanced Designer page and click **Settings**. Scroll down to the **Security** heading and check the **Restrict Access** checkbox. When this checkbox is enabled your App will:

- **Not** appear on any Homepage associated with a user that does not have access to at least one of the App's Records (via any security layer).
- **Not** allow any new Records to be created using the Onit user interface, even by those users that can see the App on their Homepage, *unless* that user is part of a User Group that has the role "Private App Initiator".

The screenshot shows the Onit App Builder interface. On the left is a sidebar with a menu. The 'Settings' option under 'App Builder' is highlighted with a red box. The main panel on the right is titled 'GENERAL' and 'APPEARANCE'. At the bottom of the main panel, the 'SECURITY' section is highlighted with a red box. Inside the 'SECURITY' section, the 'Restrict Access' checkbox is checked, and a red arrow points to it. Below it, the 'Hide emails in log' checkbox is unchecked.

Security Layer: Security Chains

This advanced layer of security enables you to grant/deny access to individual transactions based on whether they have access to a *related* transaction. That probably sounds pretty abstract, so let's consider this security layer from the vantage point of a real-world use-case.

Imagine that you have a **Contracts** app and that it contains a Field named **Country**. Also imagine that your users have been assigned a property in Onit which designates which country they work in. Finally, let's say that you need

to grant access to transactions based on which country a user works in. For example, if a particular **Contracts** transaction has a **Country** Field value of **Singapore**, Onit should limit access to this transaction to only those users who work in Singapore.

This is a good use-case for the Security Chain security layer. Sticking with the example above, here is how you could set this up:

1. Create an app named **Country-Permissions**, inside of which you'll create one transaction per possible country (e.g., one for Singapore, another for Mexico, etc.).
2. Set the **Contracts** app to be a sibling of the **Country-Permissions** app.

Note: Though its more common to create Security Chains for sibling apps, note that it's also possible to configure them for parent-child app relationships when your business use case calls for that relationship type.


3. Assign your users to be Participants of the **Country-Permissions** transactions. For instance, if Sarah works in Singapore, assign her to be a Participant on the **Singapore** transaction.

Note: It does not matter what Role type your participants are provided here. The Role type that we'll define later (when we wire up the Inherited Role for the Security Chain) will override this Role.

4. When a new transaction is created in your **Contracts** app, fire a **Find or Create Related Transaction** (FCRT) Action to create a sibling relationship between it and the appropriate transaction in the **Country-Permissions** app. That is, you would configure the FCRT transaction to get the value of the **Contract** transaction's **Country** Field and then go find the matching transaction in the **Country-Permissions** app. Once a match is found, the FCRT will relate the two transactions together.

Below is a screenshot of how your FCRT might be configured:

Find or Create Related Transaction

Name *	Country Relation	
Description		
Related Property Name *	Country Permissions (countrypermissions)	
Operation *	Upsert	
Relation Field Name *	name	
Relation Field Value *	{{country}}	
Params Liquid *	name:{{country}}	
Comment	Provide a brief description of the change you made (optional)	
		<button>Save</button>

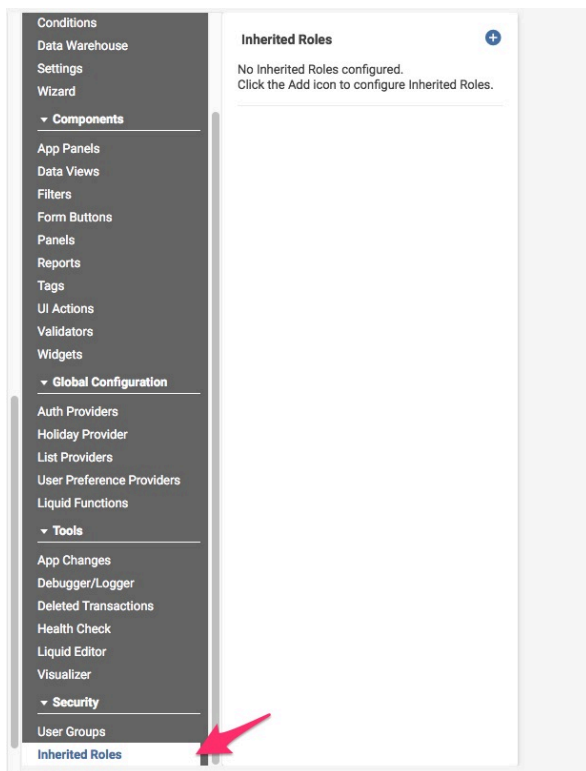
5. Finally, you'll need to set up something called an Inherited Role within your **Contracts** app. This will essentially tell Onit, "For any given **Contracts** transaction, find its related **Country-Permissions** transaction and identify all of its Participants. Grant those Participants access to the **Contracts** transaction." For example, if **Contract 123** is related to **Singapore**, then grant access to all Participants on the **Singapore** transaction.

Using Inherited Roles requires you to understand several different complex Onit concepts. While it's outside the scope of this tutorial to provide a step-by-step guide to setting all of this up, below is a brief description of how to create the Inherited Role for a Security Chain inside of an App (Step 5 from the list above).

Creating Inherited Roles

Identify which App contains the Records that you ultimately want to grant users access to. (In our example above, that would be the **Contracts** App.) Browse to this App's **Advanced Designer** page.

From the left-hand pane, select the **Inherited Roles** node.



Click **Add** and **Name** your Inherited Role whatever you like.

From the **Role** dropdown, select a Role. This controls what the users can do on any given transaction. (E.g., If the Role is limited to read-only access, that permission will filter down to the Inherited Role that you are creating).

For the **Source Apps** property, select one or more apps. In our example above, we would select the **Country-Permissions** app.

Advanced Configuration: A quick sidebar... You might be wondering why you might select *multiple* Apps here. In some cases, you'll want to chain a series of Apps together to set up your Security Chain. This can be an especially complex type of configuration, but the easiest way to explain why this may be necessary is to expand our real-world example from above. It's common to have an App whose sole purpose is to manage your environment's users and their properties. This App is usually named **User Profiles**, and normally you'll add users to be a Participant on their own **User Profiles** Record (e.g., Bob would be a Participant on the **User Profiles** Record that represented him). In this situation, you don't want to add Bob to be both a Participant on his **User Profiles** Record *and* on the **Country-Permissions** Records that identified the country he lives in -- that's too much work to do for each user. Instead, it'd be much easier to simply create a Field inside of the **User Profiles** App named **Country**, and then use that Field to identify where Bob lives. Using this approach, you could then have Onit create the following sibling relationships to pass an Inherited Roles from a **User Profiles** Record to a **Country-Permissions** Record and onto your **Contracts** Records:

- A sibling relationship from Bob's **User Profiles** Record to the appropriate **Country-Permission** Record.
- A sibling relationship from the **Country-Permission** Record to the **Contracts** Record.

Gotcha: If you are trying to provide access between a chain of three Apps related with HasMany and BelongsTo, you must configure an Inherited Role to the "grandparent" App to the "grandchild" App in the chain.

Again, this is an especially complex Security Chain set up, but it's important to know that it's an option. For this tutorial, we'll keep it simple and create a Security Chain between only two Apps (not three).

For the Inherited Role's **Mapping** property, we need to tell our app how to find the other app(s) that it's supposed to inherit roles from. In our example -- where we just need to map from the current app to a single sibling app -- this is pretty easy. Simply enter the name of the **ManyToMany** Fields that relate our two apps together (remember [sibling apps](#) must each have a **ManyToMany** Field and those Fields *must* be named the same in both apps.) For our example, we named our **ManyToMany** Fields **contracts_countriespermissions**, so we'll enter exactly that in our Inherited Role's **Mapping** property.

Inherited Role

Name *

Inherit from Country-Permissions

Role *

Approver

Source Apps *

Country-Permissions

Mapping *

contracts-countriespermissions

Comment


Provide a brief description of the change you made (optional)

Delete

Save

Note that the task of mapping between apps becomes a little more complex when creating Security Chains between more than two apps or between apps with parent-child relationships. Refer to the table below to find the appropriate **Mapping** property syntax for your use case:

Number of Apps	Parent-Child	Sibling
2	SourceApp_HasMany_FieldName.CurrentApp_BelongsTo_FieldName	ManyToMany_FieldName
3	SourceApp_HasMany_FieldName.MiddlemanApp_HasMany_FieldName.CurrentApp_BelongsTo_FieldName	SourceApp_ManyToMany_FieldName App_ManyToMany_FieldName

 **Note:** **SourceApp** refers to the App you want to inherit roles from, **MiddlemanApp** refers to the App Inherited Roles will need to pass through, and **CurrentApp** refers to the App that you want to provide/prevent access to.

Click **Ok** to save your Inherited Role.

You're done setting up an Inherited Role, but remember that there is more configuration that you'll need to perform. That is, relating transactions together and adding Participants to transactions.

Security Layer: Liquid Conditions

The final layer of security is the most granular of all, as it allows you to grant/deny access to individual elements in the Onit user interface, like Buttons and App Panels.

When creating a UI element, you can assign it a condition to determine when it should display. As a result, you can use this condition to only show the element to the appropriate users, based on user properties.

Let's look at an example. Imagine that you have a **Contracts** app, within which you've created a Button named **Close Contract**. While you do want many people to have access to the **Contracts** transactions, you only want a subset of those users to have access to click the **Close Contract** button. As a result, you could assign this Button a Condition which limits who can see it.

Setting up this layer of security assumes that you have a way to assign properties to users. As mentioned above, Onit provides a clean and simple way to do this, which involves creating a special app (usually named **User Profiles**). Within this app, you'll create one transaction per user, and then you'll use Field values to assign properties to each user. If necessary, you can also create a separate, related app to track group properties, so that you can simply assign users to groups and then assign properties at the group level.

Though explaining the details of user/group properties is outside the scope of this tutorial, we can cover how to leverage this type of set up to conditionally assign security to user interface elements.

Sticking with the **Contracts** app example from above, let's say that we have a checkbox Field at the group-level named **Can Close Contracts**. If so, then we could create a Condition that checks this property for the current user. We could then assign this Condition to our **Close Contracts** Button, which would control whether it is hidden/displayed for any given user.

For example, we could create a Condition that uses the following Liquid:

```
{% assign my_user = current_user | user_preferences_for_user %}{% if my_user.group.can_assign_vendors == true %}true{% endif %}
```

This Liquid does the following:

- Creates a new variable named **my_user**, which stores the user preferences object for the user that is currently logged into Onit.
- Leverages **my_user** to identify which group the current user belongs to.
- Checks to see if the group in question has its **Can Close Contracts** checkbox checked.
- If the checkbox is checked, a value of **true** is returned.

Other Things to Consider

In addition to the security layers explained above, the following is a hodgepodge of other security-related Onit settings to be aware of:

- **Tags:** Remember that both Roles and Fields are assigned to Tabs within the **Wizard**. As a result, you can show/hide Fields by putting them inside of a Tab that is restricted to a certain Role, (although a Field whose **Show on Dashboard** property is selected will still be viewable by anyone who has access to the Record on the dashboard). Keep this in mind when configuring any of the security layers mentioned above.
- **Hiding Apps:** If you are not using Onit suites, then your users are shown a list of all possible Apps when they first log into Onit (on their Homepage). In some situations, you may want to hide Apps from this list. To do so, you have the following settings available:
 - **Active:** Within the **Wizard**, on the **General** tab, the **Active** checkbox determines whether the App appears for general users on their Homepage. If you unselect this checkbox, then the App will only be visible to users that have been assigned the Admin Role of **System Administrator** or **App Creator**.
 - **Restrict Access:** Within an App's **Advanced Designer** page, you can mark the App as having "restricted access" by selecting the **Settings** node from the left-hand pane, clicking the **Restrict Access** checkbox, and clicking **OK**. When this setting is enabled:
 - It will not appear on any Homepage associated with a user that does not have access to at least one of the App's Records (via any security layer).
 - It will not allow any new Records to be created using the Onit user interface, even by those users that can see the App on their Homepage, *unless* that user is part of a User Group that has the role "Private App Initiator".

User Group

Name * UserGroup1

Roles *

Comment

Approver

App Administrator

Legal Approver

Private App Initiator

Requester

Viewer

In Summary

And that's it! All simple stuff, right?

Not exactly. Security is a critical component of any system, and one that should be carefully designed, configured, tested, and maintained. Hopefully this tutorial has provided a quick and dirty overview of the security tools that you have available in Onit.

To put everything that you've learned here into action, we recommend starting small and simple. Creating a test user, pick a basic layer of security mentioned above, define your use-case upfront, and then set out to achieve it. As you master one security layer, move on to the next one. Before you know it, you'll have a robust security design that accomplishes what you need. Good luck!