Apache NiFi 3

Apache NiFi Security

Date of Publish: 2018-08-13

Contents

User Authentication

NiFi supports user authentication via client certificates, via username/password, via Apache Knox, or via http://openid.net/connect.

Username/password authentication is performed by a 'Login Identity Provider'. The Login Identity Provider is a pluggable mechanism for authenticating users via their username/password. Which Login Identity Provider to use is configured in the nifi.properties file. Currently NiFi offers username/password with Login Identity Providers options for LDAP and Kerberos.

The nifi.login.identity.provider.configuration.file property specifies the configuration file for Login Identity Providers. The nifi.security.user.login.identity.provider property indicates which of the configured Login Identity Provider should be used. By default, this property is not configured meaning that username/password must be explicitly enabled.

During OpenId Connect authentication, NiFi will redirect users to login with the Provider before returning to NiFi. NiFi will then call the Provider to obtain the user identity.

During Apache Knox authentication, NiFi will redirect users to login with Apache Knox before returning to NiFi. NiFi will verify the Apache Knox token during authentication.

Note: NiFi can only be configured for username/password, OpenId Connect, or Apache Knox at a given time. It does not support running each of these concurrently. NiFi will require client certificates for authenticating users over HTTPS if none of these are configured.

A secured instance of NiFi cannot be accessed anonymously unless configured to use an LDAP or Kerberos Login Identity Provider, which in turn must be configured to explicitly allow anonymous access. Anonymous access is not currently possible by the default FileAuthorizer.

Note: NiFi does not perform user authentication over HTTP. Using HTTP, all users will be granted all roles.

Lightweight Directory Access Protocol (LDAP)

Below is an example and description of configuring a Login Identity Provider that integrates with a Directory Server to authenticate users.

```
<provider>
    <identifier>ldap-provider</identifier>
    <class>org.apache.nifi.ldap.LdapProvider</class>
    <property name="Authentication Strategy">START_TLS</property>

    <property name="Manager DN"></property>
    <property name="Manager Password"></property>

    <property name="TLS - Keystore"></property>
    <property name="TLS - Keystore Password"></property>
    <property name="TLS - Keystore Type"></property>
    <property name="TLS - Truststore"></property>
    <property name="TLS - Truststore Password"></property>
    <property name="TLS - Truststore Type"></property>
    <property name="TLS - Client Auth"></property>
    <property name="TLS - Protocol"></property>
    <property name="TLS - Shutdown Gracefully"></property>

    <property name="Referral Strategy">FOLLOW</property>
    <property name="Connect Timeout">10 secs</property>
    <property name="Read Timeout">10 secs</property>
    <property name="Url"></property>
    <property name="User Search Base"></property>
    <property name="User Search Filter"></property>

    <property name="Identity Strategy">USE_DN</property>
    <property name="Authentication Expiration">12 hours</property>
</provider>
```

With this configuration, username/password authentication can be enabled by referencing this provider in nifi.properties.

```
nifi.security.user.login.identity.provider=ldap-provider
```

Property Name	**Description**
Authentication Expiration	The duration of how long the user authentication is valid for. If the user never logs out, they will be required to log back in following this duration.
Authentication Strategy	How the connection to the LDAP server is authenticated. Possible values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS.
Manager DN	The DN of the manager that is used to bind to the LDAP server to search for users.
Manager Password	The password of the manager that is used to bind to the LDAP server to search for users.
TLS - Keystore	Path to the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
TLS - Keystore Password	Password for the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
TLS - Keystore Type	Type of the Keystore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
TLS - Truststore	Path to the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
TLS - Truststore Password	Password for the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
TLS - Truststore Type	Type of the Truststore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
TLS - Client Auth	Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, NONE.
TLS - Protocol	Protocol to use when connecting to LDAP using LDAPS or START_TLS. (i.e. TLS, TLSv1.1, TLSv1.2, etc).
TLS - Shutdown Gracefully	Specifies whether the TLS should be shut down gracefully before the target context is closed. Defaults to false.
Referral Strategy	Strategy for handling referrals. Possible values are FOLLOW, IGNORE, THROW.
Connect Timeout	Duration of connect timeout. (i.e. 10 secs).
Read Timeout	Duration of read timeout. (i.e. 10 secs).
Url	Space-separated list of URLs of the LDAP servers (i.e. ldap://<hostname>:<port>).

Property Name	Description
User Search Base	Base DN for searching for users (i.e. CN=Users,DC=example,DC=com).
User Search Filter	Filter for searching for users against the 'User Search Base'. (i.e. sAMAccountName={0}). The user specified name is inserted into '{0}'.
Identity Strategy	Strategy to identify users. Possible values are USE_DN and USE_USERNAME. The default functionality if this property is missing is USE_DN in order to retain backward compatibility. USE_DN will use the full DN of the user entry if possible. USE_USERNAME will use the username the user logged in with.

Kerberos

Below is an example and description of configuring a Login Identity Provider that integrates with a Kerberos Key Distribution Center (KDC) to authenticate users.

```
<provider>
    <identifier>kerberos-provider</identifier>
    <class>org.apache.nifi.kerberos.KerberosProvider</class>
    <property name="Default Realm">NIFI.APACHE.ORG</property>
    <property name="Kerberos Config File">/etc/krb5.conf</property>
    <property name="Authentication Expiration">12 hours</property>
</provider>
```

With this configuration, username/password authentication can be enabled by referencing this provider in nifi.properties.

```
nifi.security.user.login.identity.provider=kerberos-provider
```

Property Name	Description
Authentication Expiration	The duration of how long the user authentication is valid for. If the user never logs out, they will be required to log back in following this duration.
Default Realm	Default realm to provide when user enters incomplete user principal (i.e. NIFI.APACHE.ORG).
Kerberos Config File	Absolute path to Kerberos client configuration file.

OpenId Connect

To enable authentication via OpenId Connect the following properties must be configured in nifi.properties.

Property Name	Description
nifi.security.user.oidc.preferred.jwsalgorithm	The preferred algorithm for for validating identity tokens. If this value is blank, it will default to 'RS256' which is required to be supported by the OpenId Connect Provider according to the specification. If this value is 'HS256', 'HS384', or 'HS512', NiFi will attempt to validate HMAC protected tokens using the specified client secret. If this value is 'none', NiFi will attempt to validate unsecured/plain tokens. Other values for this algorithm will attempt to parse as an RSA or EC algorithm to be used in conjunction with the JSON Web Key (JWK) provided through the jwks_uri in the metadata found at the discovery URL.
nifi.security.user.oidc.discovery.url	The discovery URL for the desired OpenId Connect Provider (http://openid.net/specs/openid-connect-discovery-1_0.html).
nifi.security.user.oidc.connect.timeout	Connect timeout when communicating with the OpenId Connect Provider.
nifi.security.user.oidc.read.timeout	Read timeout when communicating with the OpenId Connect Provider.
nifi.security.user.oidc.client.id	The client id for NiFi after registration with the OpenId Connect Provider.
nifi.security.user.oidc.client.secret	The client secret for NiFi after registration with the OpenId Connect Provider.

Apache Knox

To enable authentication via Apache Knox the following properties must be configured in nifi.properties.

Property Name	Description
nifi.security.user.knox.audiences	Optional. A comma separate listed of allowed audiences. If set, the audience in the token must be present in this listing. The audience that is populated in the token can be configured in Knox.
nifi.security.user.knox.url	The URL for the Apache Knox log in page.
nifi.security.user.knox.publicKey	The path to the Apache Knox public key that will be used to verify the signatures of the authentication tokens in the HTTP Cookie.
nifi.security.user.knox.cookieName	The name of the HTTP Cookie that Apache Knox will generate after successful log in.

Multi-Tenant Authorization

After you have configured NiFi to run securely and with an authentication mechanism, you must configure who has access to the system, and the level of their access. You can do this using 'multi-tenant authorization'. Multi-tenant authorization enables multiple groups of users (tenants) to command, control, and observe different parts of the dataflow, with varying levels of authorization. When an authenticated user attempts to view or modify a NiFi resource, the system checks whether the user has privileges to perform that action. These privileges are defined by policies that you can apply system-wide or to individual components.

Authorizer Configuration

An 'authorizer' grants users the privileges to manage users and policies by creating preliminary authorizations at startup.

Authorizers are configured using two properties in the 'nifi.properties' file:

- The nifi.authorizer.configuration.file property specifies the configuration file where authorizers are defined. By default, the 'authorizers.xml' file located in the root installation conf directory is selected.
- The nifi.security.user.authorizer property indicates which of the configured authorizers in the 'authorizers.xml' file to use.

Authorizers.xml Setup

The 'authorizers.xml' file is used to define and configure available authorizers. The default authorizer is the StandardManagedAuthorizer. The managed authorizer is comprised of a UserGroupProvider and a AccessPolicyProvider. The users, group, and access policies will be loaded and optionally configured through these providers. The managed authorizer will make all access decisions based on these provided users, groups, and access policies.

During startup there is a check to ensure that there are no two users/groups with the same identity/name. This check is executed regardless of the configured implementation. This is necessary because this is how users/groups are identified and authorized during access decisions.

The default UserGroupProvider is the FileUserGroupProvider, however, you can develop additional UserGroupProviders as extensions. The FileUserGroupProvider has the following properties:

- Users File - The file where the FileUserGroupProvider stores users and groups. By default, the 'users.xml' in the 'conf' directory is chosen.
- Legacy Authorized Users File - The full path to an existing authorized-users.xml that will be automatically be used to load the users and groups into the Users File.
- Initial User Identity - The identity of a users and systems to seed the Users File. The name of each property must be unique, for example: "Initial User Identity A", "Initial User Identity B", "Initial User Identity C" or "Initial User Identity 1", "Initial User Identity 2", "Initial User Identity 3"

Another option for the UserGroupProvider is the LdapUserGroupProvider. By default, this option is commented out but can be configured in lieu of the FileUserGroupProvider. This will sync users and groups from a directory server and will present them in NiFi UI in read only form. The LdapUserGroupProvider has the following properties:

- Authentication Strategy - How the connection to the LDAP server is authenticated. Possible values are ANONYMOUS, SIMPLE, LDAPS, or START_TLS
- Manager DN - The DN of the manager that is used to bind to the LDAP server to search for users.
- Manager Password - The password of the manager that is used to bind to the LDAP server to search for users.
- TLS - Keystore - Path to the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Keystore Password - Password for the Keystore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Keystore Type - Type of the Keystore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
- TLS - Truststore - Path to the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Truststore Password - Password for the Truststore that is used when connecting to LDAP using LDAPS or START_TLS.
- TLS - Truststore Type - Type of the Truststore that is used when connecting to LDAP using LDAPS or START_TLS (i.e. JKS or PKCS12).
- TLS - Client Auth - Client authentication policy when connecting to LDAP using LDAPS or START_TLS. Possible values are REQUIRED, WANT, NONE.
- TLS - Protocol - Protocol to use when connecting to LDAP using LDAPS or START_TLS. (i.e. TLS, TLSv1.1, TLSv1.2, etc).

- TLS - Shutdown Gracefully - Specifies whether the TLS should be shut down gracefully before the target context is closed. Defaults to false.
- Referral Strategy - Strategy for handling referrals. Possible values are FOLLOW, IGNORE, THROW.
- Connect Timeout - Duration of connect timeout. (i.e. 10 secs).
- Read Timeout - Duration of read timeout. (i.e. 10 secs).
- Url - Space-separated list of URLs of the LDAP servers (i.e. ldap://<hostname>:<port>).
- Page Size - Sets the page size when retrieving users and groups. If not specified, no paging is performed.
- Sync Interval - Duration of time between syncing users and groups. (i.e. 30 mins). Minimum allowable value is 10 secs.
- User Search Base - Base DN for searching for users (i.e. ou=users,o=nifi). Required to search users.
- User Object Class - Object class for identifying users (i.e. person). Required if searching users.
- User Search Scope - Search scope for searching users (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching users.
- User Search Filter - Filter for searching for users against the 'User Search Base' (i.e. (memberof=cn=team1,ou=groups,o=nifi)). Optional.
- User Identity Attribute - Attribute to use to extract user identity (i.e. cn). Optional. If not set, the entire DN is used.
- User Group Name Attribute - Attribute to use to define group membership (i.e. memberof). Optional. If not set group membership will not be calculated through the users. Will rely on group membership being defined through 'Group Member Attribute' if set. The value of this property is the name of the attribute in the user ldap entry that associates them with a group. The value of that user attribute could be a dn or group name for instance. What value is expected is configured in the 'User Group Name Attribute - Referenced Group Attribute'.
- User Group Name Attribute - Referenced Group Attribute - If blank, the value of the attribute defined in 'User Group Name Attribute' is expected to be the full dn of the group. If not blank, this property will define the attribute of the group ldap entry that the value of the attribute defined in 'User Group Name Attribute' is referencing (i.e. name). Use of this property requires that 'Group Search Base' is also configured.
- Group Search Base - Base DN for searching for groups (i.e. ou=groups,o=nifi). Required to search groups.
- Group Object Class - Object class for identifying groups (i.e. groupOfNames). Required if searching groups.
- Group Search Scope - Search scope for searching groups (ONE_LEVEL, OBJECT, or SUBTREE). Required if searching groups.
- Group Search Filter - Filter for searching for groups against the 'Group Search Base'. Optional.
- Group Name Attribute - Attribute to use to extract group name (i.e. cn). Optional. If not set, the entire DN is used.
- Group Member Attribute - Attribute to use to define group membership (i.e. member). Optional. If not set group membership will not be calculated through the groups. Will rely on group membership being defined through 'User Group Name Attribute' if set. The value of this property is the name of the attribute in the group ldap entry that associates them with a user. The value of that group attribute could be a dn or memberUid for instance. What value is expected is configured in the 'Group Member Attribute - Referenced User Attribute'. (i.e. member: cn=User 1,ou=users,o=nifi vs. memberUid: user1)
- Group Member Attribute - Referenced User Attribute - If blank, the value of the attribute defined in 'Group Member Attribute' is expected to be the full dn of the user. If not blank, this property will define the attribute of the user ldap entry that the value of the attribute defined in 'Group Member Attribute' is referencing (i.e. uid). Use of this property requires that 'User Search Base' is also configured. (i.e. member: cn=User 1,ou=users,o=nifi vs. memberUid: user1)

Another option for the UserGroupProvider are composite implementations. This means that multiple sources/implementations can be configured and composed. For instance, an admin can configure users/groups to be loaded from a file and a directory server. There are two composite implementations, one that supports multiple UserGroupProviders and one that supports multiple UserGroupProviders and a single configurable UserGroupProvider.

The CompositeUserGroupProvider will provide support for retrieving users and groups from multiple sources. The CompositeUserGroupProvider has the following properties:

- User Group Provider - The identifier of user group providers to load from. The name of each property must be unique, for example: "User Group Provider A", "User Group Provider B", "User Group Provider C" or "User Group Provider 1", "User Group Provider 2", "User Group Provider 3"

The CompositeConfigurableUserGroupProvider will provide support for retrieving users and groups from multiple sources. Additionally, a single configurable user group provider is required. Users from the configurable user group provider are configurable, however users loaded from one of the User Group Provider [unique key] will not be. The CompositeConfigurableUserGroupProvider has the following properties:

- Configurable User Group Provider - A configurable user group provider.
- User Group Provider - The identifier of user group providers to load from. The name of each property must be unique, for example: "User Group Provider A", "User Group Provider B", "User Group Provider C" or "User Group Provider 1", "User Group Provider 2", "User Group Provider 3"

The default AccessPolicyProvider is the FileAccessPolicyProvider, however, you can develop additional AccessPolicyProvider as extensions. The FileAccessPolicyProvider has the following properties:

- User Group Provider - The identifier for an User Group Provider defined above that will be used to access users and groups for use in the managed access policies.
- Authorizations File - The file where the FileAccessPolicyProvider will store policies.
- Initial Admin Identity - The identity of an initial admin user that will be granted access to the UI and given the ability to create additional users, groups, and policies. The value of this property could be a DN when using certificates or LDAP, or a Kerberos principal. This property will only be used when there are no other policies defined. If this property is specified then a Legacy Authorized Users File can not be specified.
- Legacy Authorized Users File - The full path to an existing authorized-users.xml that will be automatically converted to the new authorizations model. If this property is specified then an Initial Admin Identity can not be specified, and this property will only be used when there are no other users, groups, and policies defined.
- Node Identity - The identity of a NiFi cluster node. When clustered, a property for each node should be defined, so that every node knows about every other node. If not clustered these properties can be ignored. The name of each property must be unique, for example for a three node cluster: "Node Identity A", "Node Identity B", "Node Identity C" or "Node Identity 1", "Node Identity 2", "Node Identity 3"

The identities configured in the Initial Admin Identity, the Node Identity properties, or discovered in a Legacy Authorized Users File must be available in the configured User Group Provider.

The default authorizer is the StandardManagedAuthorizer, however, you can develop additional authorizers as extensions. The StandardManagedAuthorizer has the following properties:

- Access Policy Provider - The identifier for an Access Policy Provider defined above.

The FileAuthorizer has been replaced with the more granular StandardManagedAuthorizer approach described above. However, it is still available for backwards compatibility reasons. The FileAuthorizer has the following properties.

- Authorizations File - The file where the FileAuthorizer stores policies. By default, the 'authorizations.xml' in the 'conf' directory is chosen.
- Users File - The file where the FileAuthorizer stores users and groups. By default, the 'users.xml' in the 'conf' directory is chosen.
- Initial Admin Identity - The identity of an initial admin user that is granted access to the UI and given the ability to create additional users, groups, and policies. This property is only used when there are no other users, groups, and policies defined.
- Legacy Authorized Users File - The full path to an existing authorized-users.xml that is automatically converted to the multi-tenant authorization model. This property is only used when there are no other users, groups, and policies defined.
- Node Identity - The identity of a NiFi cluster node. When clustered, a property for each node should be defined, so that every node knows about every other node. If not clustered, these properties can be ignored.

Initial Admin Identity (New NiFi Instance)

If you are setting up a secured NiFi instance for the first time, you must manually designate an "Initial Admin Identity" in the 'authorizers.xml' file. This initial admin user is granted access to the UI and given the ability to create additional users, groups, and policies. The value of this property could be a DN (when using certificates or LDAP) or a Kerberos principal. If you are the NiFi administrator, add yourself as the "Initial Admin Identity".

Here is an example LDAP entry using the name John Smith:

```
<authorizers>
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Initial User Identity 1">cn=John
 Smith,ou=people,dc=example,dc=com</property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">cn=John
 Smith,ou=people,dc=example,dc=com</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity 1"></property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

Here is an example Kerberos entry using the name John Smith and realm NIFI.APACHE.ORG:

```
<authorizers>
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Initial User Identity 1">johnsmith@NIFI.APACHE.ORG</
property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">johnsmith@NIFI.APACHE.ORG</
property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity 1"></property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

After you have edited and saved the 'authorizers.xml' file, restart NiFi. The "Initial Admin Identity" user and administrative policies are added to the 'users.xml' and 'authorizations.xml' files during restart. Once NiFi starts, the "Initial Admin Identity" user is able to access the UI and begin managing users, groups, and policies.

> **Note:** For a brand new secure flow, providing the "Initial Admin Identity" gives that user access to get into the UI and to manage users, groups and policies. But if that user wants to start modifying the flow, they need to grant themselves policies for the root process group. The system is unable to do this automatically because in a new flow the UUID of the root process group is not permanent until the flow.xml.gz is generated. If the NiFi instance is an upgrade from an existing flow.xml.gz or a 1.x instance going from unsecure to secure, then the "Initial Admin Identity" user is automatically given the privileges to modify the flow.

Here is an example loading users and groups from LDAP. Group membership will be driven through the member attribute of each group. Authorization will still use file based access policies:

```
dn: cn=User 1,ou=users,o=nifi
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
objectClass: top
cn: User 1
sn: User1
uid: user1

dn: cn=User 2,ou=users,o=nifi
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
objectClass: top
cn: User 2
sn: User2
uid: user2

dn: cn=admins,ou=groups,o=nifi
objectClass: groupOfNames
objectClass: top
cn: admins
member: cn=User 1,ou=users,o=nifi
member: cn=User 2,ou=users,o=nifi

<authorizers>
    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>
        <class>org.apache.nifi.ldap.tenants.LdapUserGroupProvider</class>
        <property name="Authentication Strategy">ANONYMOUS</property>

        <property name="Manager DN"></property>
        <property name="Manager Password"></property>

        <property name="TLS - Keystore"></property>
        <property name="TLS - Keystore Password"></property>
        <property name="TLS - Keystore Type"></property>
        <property name="TLS - Truststore"></property>
        <property name="TLS - Truststore Password"></property>
        <property name="TLS - Truststore Type"></property>
        <property name="TLS - Client Auth"></property>
        <property name="TLS - Protocol"></property>
        <property name="TLS - Shutdown Gracefully"></property>

        <property name="Referral Strategy">FOLLOW</property>
        <property name="Connect Timeout">10 secs</property>
        <property name="Read Timeout">10 secs</property>

        <property name="Url">ldap://localhost:10389</property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base">ou=users,o=nifi</property>
        <property name="User Object Class">person</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute">cn</property>
        <property name="User Group Name Attribute"></property>
        <property name="User Group Name Attribute - Referenced Group
 Attribute"></property>

        <property name="Group Search Base">ou=groups,o=nifi</property>
        <property name="Group Object Class">groupOfNames</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute">cn</property>
        <property name="Group Member Attribute">member</property>
        <property name="Group Member Attribute - Referenced User
 Attribute"></property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">ldap-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">John Smith</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity 1"></property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

The 'Initial Admin Identity' value would have loaded from the cn from John Smith's entry based on the 'User Identity Attribute' value.

Here is an example loading users and groups from LDAP. Group membership will be driven through the member attribute of each group. Authorization will still use file based access policies:

```
dn: uid=User 1,ou=Users,dc=local
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
```

```
uid: user1
cn: User 1

dn: uid=User 2,ou=Users,dc=local
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: user2
cn: User 2

dn: cn=Managers,ou=Groups,dc=local
objectClass: posixGroup
cn: Managers
memberUid: user1
memberUid: user2

<authorizers>
    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>
        <class>org.apache.nifi.ldap.tenants.LdapUserGroupProvider</class>
        <property name="Authentication Strategy">ANONYMOUS</property>

        <property name="Manager DN"></property>
        <property name="Manager Password"></property>

        <property name="TLS - Keystore"></property>
        <property name="TLS - Keystore Password"></property>
        <property name="TLS - Keystore Type"></property>
        <property name="TLS - Truststore"></property>
        <property name="TLS - Truststore Password"></property>
        <property name="TLS - Truststore Type"></property>
        <property name="TLS - Client Auth"></property>
        <property name="TLS - Protocol"></property>
        <property name="TLS - Shutdown Gracefully"></property>

        <property name="Referral Strategy">FOLLOW</property>
        <property name="Connect Timeout">10 secs</property>
        <property name="Read Timeout">10 secs</property>

        <property name="Url">ldap://localhost:10389</property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base">ou=Groups,dc=local</property>
        <property name="User Object Class">posixAccount</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute">cn</property>
        <property name="User Group Name Attribute"></property>
        <property name="User Group Name Attribute - Referenced Group
 Attribute"></property>

        <property name="Group Search Base">ou=Groups,dc=local</property>
        <property name="Group Object Class">posixGroup</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute">cn</property>
        <property name="Group Member Attribute">memberUid</property>
        <property name="Group Member Attribute - Referenced User
 Attribute">uid</property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">ldap-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">John Smith</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity 1"></property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

Here is an example composite implementation loading users and groups from LDAP and a local file. Group membership will be driven through the member attribute of each group. The users from LDAP will be read only while the users loaded from the file will be configurable in UI.

```
<authorizers>
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Initial User Identity 1">cn=nifi-
node1,ou=servers,dc=example,dc=com</property>
        <property name="Initial User Identity 2">cn=nifi-
node2,ou=servers,dc=example,dc=com</property>
    </userGroupProvider>
    <userGroupProvider>
        <identifier>ldap-user-group-provider</identifier>
        <class>org.apache.nifi.ldap.tenants.LdapUserGroupProvider</class>
        <property name="Authentication Strategy">ANONYMOUS</property>

        <property name="Manager DN"></property>
        <property name="Manager Password"></property>

        <property name="TLS - Keystore"></property>
        <property name="TLS - Keystore Password"></property>
        <property name="TLS - Keystore Type"></property>
        <property name="TLS - Truststore"></property>
        <property name="TLS - Truststore Password"></property>
        <property name="TLS - Truststore Type"></property>
        <property name="TLS - Client Auth"></property>
        <property name="TLS - Protocol"></property>
        <property name="TLS - Shutdown Gracefully"></property>

        <property name="Referral Strategy">FOLLOW</property>
        <property name="Connect Timeout">10 secs</property>
        <property name="Read Timeout">10 secs</property>

        <property name="Url">ldap://localhost:10389</property>
        <property name="Page Size"></property>
        <property name="Sync Interval">30 mins</property>

        <property name="User Search Base">ou=users,o=nifi</property>
        <property name="User Object Class">person</property>
        <property name="User Search Scope">ONE_LEVEL</property>
        <property name="User Search Filter"></property>
        <property name="User Identity Attribute">cn</property>
        <property name="User Group Name Attribute"></property>
        <property name="User Group Name Attribute - Referenced Group
 Attribute"></property>

        <property name="Group Search Base">ou=groups,o=nifi</property>
        <property name="Group Object Class">groupOfNames</property>
        <property name="Group Search Scope">ONE_LEVEL</property>
        <property name="Group Search Filter"></property>
        <property name="Group Name Attribute">cn</property>
        <property name="Group Member Attribute">member</property>
        <property name="Group Member Attribute - Referenced User
 Attribute"></property>
    </userGroupProvider>
    <userGroupProvider>
        <identifier>composite-user-group-provider</identifier>

 <class>org.apache.nifi.authorization.CompositeConfigurableUserGroupProvider</
class>
        <property name="Configurable User Group Provider">file-user-group-
provider</property>
        <property name="User Group Provider 1">ldap-user-group-provider</
property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">composite-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">John Smith</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity 1">cn=nifi-
node1,ou=servers,dc=example,dc=com</property>
        <property name="Node Identity 2">cn=nifi-
node2,ou=servers,dc=example,dc=com</property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

In this example, the users and groups are loaded from LDAP but the servers are managed in a local file. The 'Initial Admin Identity' value came from an attribute in a LDAP entry based on the 'User Identity Attribute'. The 'Node Identity' values are established in the local file using the 'Initial User Identity' properties.

Legacy Authorized Users (NiFi Instance Upgrade)

If you are upgrading from a 0.x NiFi instance, you can convert your previously configured users and roles to the multi-tenant authorization model. In the 'authorizers.xml' file, specify the location of your existing 'authorized-users.xml' file in the "Legacy Authorized Users File" property.

Here is an example entry:

```
<authorizers>
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File">/Users/johnsmith/
config_files/authorized-users.xml</property>

        <property name="Initial User Identity 1"></property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity"></property>
        <property name="Legacy Authorized Users File">/Users/johnsmith/
config_files/authorized-users.xml</property>

        <property name="Node Identity 1"></property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

After you have edited and saved the 'authorizers.xml' file, restart NiFi. Users and roles from the 'authorized-users.xml' file are converted and added as identities and policies in the 'users.xml' and 'authorizations.xml' files. Once the application starts, users who previously had a legacy Administrator role can access the UI and begin managing users, groups, and policies.

The following tables summarize the global and component policies assigned to each legacy role if the NiFi instance has an existing 'flow.xml.gz':

Global Access Policies

	Admin	DFM	Monitor	Provenance	NiFi	Proxy
view the UI	*	*	*			
access the controller - view	*	*	*		*	
access the controller - modify		*				
query provenance				*		
access restricted components		*				
access all policies - view	*					

	Admin	DFM	Monitor	Provenance	NiFi	Proxy
access all policies - modify	*					
access users/user groups - view	*					
access users/user groups - modify	*					
retrieve site-to-site details					*	
view system diagnostics		*	*			
proxy user requests						*
access counters						

Component Access Policies on the Root Process Group

	Admin	DFM	Monitor	Provenance	NiFi	Proxy
view the component	*	*	*			
modify the component		*				
view the data		*		*		*
modify the data		*				*
view provenance				*		

Note: NiFi fails to restart if values exist for both the "Initial Admin Identity" and "Legacy Authorized Users File" properties. You can specify only one of these values to initialize authorizations.

Note: Do not manually edit the 'authorizations.xml' file. Create authorizations only during initial setup and afterwards using the NiFi UI.

Cluster Node Identities

If you are running NiFi in a clustered environment, you must specify the identities for each node. The authorization policies required for the nodes to communicate are created during startup.

For example, if you are setting up a 2 node cluster with the following DNs for each node:

```
cn=nifi-1,ou=people,dc=example,dc=com
cn=nifi-2,ou=people,dc=example,dc=com
```

```
<authorizers>
    <userGroupProvider>
        <identifier>file-user-group-provider</identifier>
        <class>org.apache.nifi.authorization.FileUserGroupProvider</class>
        <property name="Users File">./conf/users.xml</property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Initial User Identity 1">johnsmith@NIFI.APACHE.ORG</
property>
```

```
        <property name="Initial User Identity
 2">cn=nifi-1,ou=people,dc=example,dc=com</property>
        <property name="Initial User Identity
 3">cn=nifi-2,ou=people,dc=example,dc=com</property>
    </userGroupProvider>
    <accessPolicyProvider>
        <identifier>file-access-policy-provider</identifier>
        <class>org.apache.nifi.authorization.FileAccessPolicyProvider</
class>
        <property name="User Group Provider">file-user-group-provider</
property>
        <property name="Authorizations File">./conf/authorizations.xml</
property>
        <property name="Initial Admin Identity">johnsmith@NIFI.APACHE.ORG</
property>
        <property name="Legacy Authorized Users File"></property>

        <property name="Node Identity
 1">cn=nifi-1,ou=people,dc=example,dc=com</property>
        <property name="Node Identity
 2">cn=nifi-2,ou=people,dc=example,dc=com</property>
    </accessPolicyProvider>
    <authorizer>
        <identifier>managed-authorizer</identifier>
        <class>org.apache.nifi.authorization.StandardManagedAuthorizer</
class>
        <property name="Access Policy Provider">file-access-policy-
provider</property>
    </authorizer>
</authorizers>
```

Note: In a cluster, all nodes must have the same 'authorizations.xml' and 'users.xml'. The only exception is if a node has empty 'authorizations.xml' and 'user.xml' files prior to joining the cluster. In this scenario, the node inherits them from the cluster during startup.

Now that initial authorizations have been created, additional users, groups and authorizations can be created and managed in the NiFi UI.

Configuring Users & Access Policies

Depending on the capabilities of the configured UserGroupProvider and AccessPolicyProvider the users, groups, and policies will be configurable in the UI. If the extensions are not configurable the users, groups, and policies will read-only in the UI. If the configured authorizer does not use UserGroupProvider and AccessPolicyProvider the users and policies may or may not be visible and configurable in the UI based on the underlying implementation.

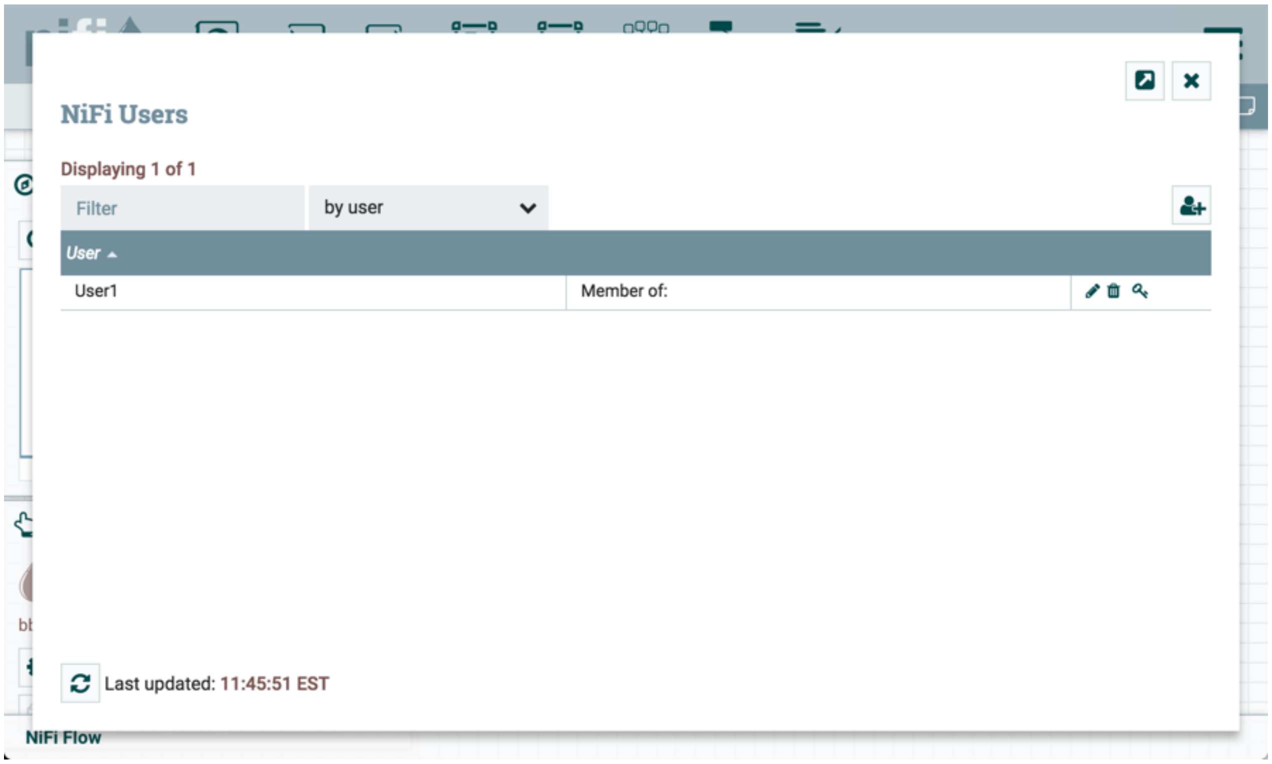
This section assumes the users, groups, and policies are configurable in the UI and describes:

- How to create users and groups
- How access policies are used to define authorizations
- How to view policies that are set on a user
- How to configure access policies by walking through specific examples

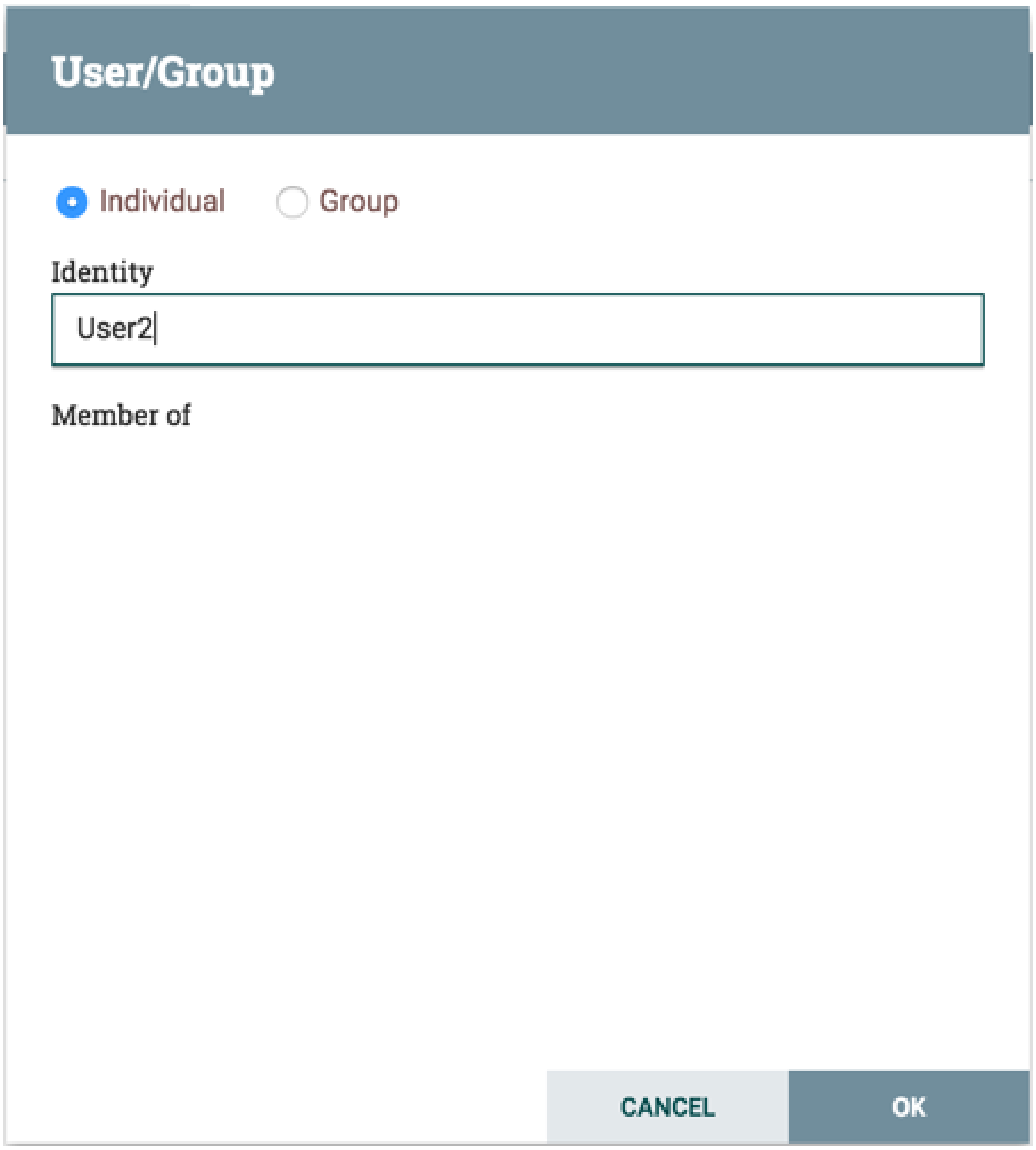
Note: Instructions requiring interaction with the UI assume the application is being accessed by User1, a user with administrator privileges, such as the "Initial Admin Identity" user or a converted legacy admin user (see Authorizers.xml Setup).

Creating Users and Groups

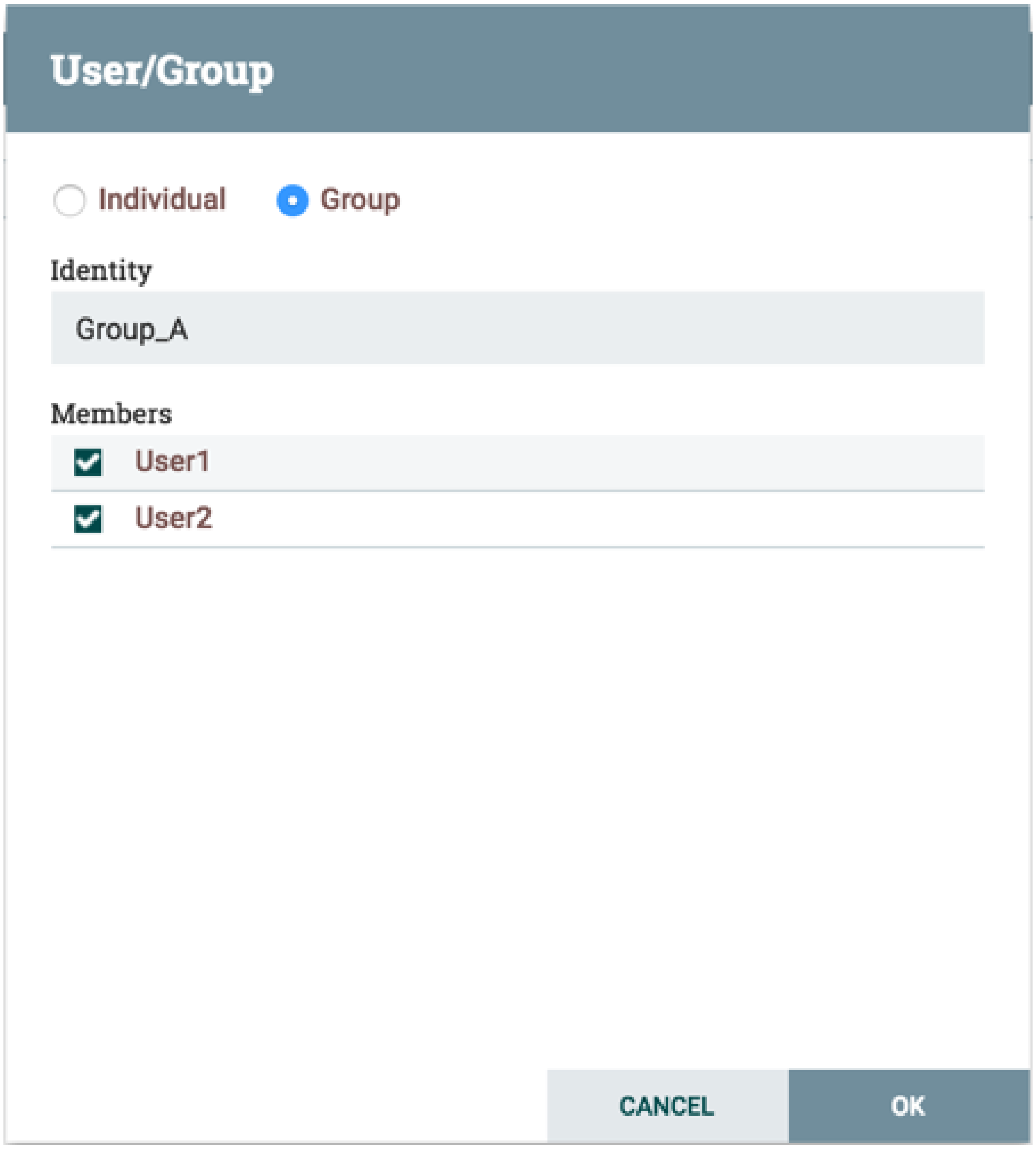
From the UI, select "Users" from the Global Menu. This opens a dialog to create and manage users and groups.



Click the Add icon ( ).
To create a user, enter the 'Identity' information relevant to the authentication method chosen to secure your NiFi instance. Click OK.



To create a group, select the "Group" radio button, enter the name of the group and select the users to be included in the group. Click OK.



Identity: Group_A

Members:
- [✓] User1
- [✓] User2

Access Policies

You can manage the ability for users and groups to view or modify NiFi resources using 'access policies'. There are two types of access policies that can be applied to a resource:

- View - If a view policy is created for a resource, only the users or groups that are added to that policy are able to see the details of that resource.
- Modify - If a resource has a modify policy, only the users or groups that are added to that policy can change the configuration of that resource.

You can create and apply access policies on both global and component levels.

Global Access Policies

Global access policies govern the following system level authorizations:

Policy	Privilege	Global Menu Selection	Resource Descriptor
view the UI	Allow users to view the UI	N/A	/flow
access the controller	Allows users to view/modify the controller including Reporting Tasks, Controller Services, and Nodes in the Cluster	Controller Settings	/controller
query provenance	Allows users to submit a Provenance Search and request Event Lineage	Data Provenance	/provenance
access restricted components	Allows users to create/modify restricted components assuming other permissions are sufficient. The restricted components may indicate which specific permissions are required. Permissions can be granted for specific restrictions or be granted regardless of restrictions. If permission is granted regardless of restrictions, the user can create/modify all restricted components.	N/A	/restricted-components
access all policies	Allows users to view/modify the policies for all components	Policies	/policies
access users/user groups	Allows users to view/modify the users and user groups	Users	/tenants
retrieve site-to-site details	Allows other NiFi instances to retrieve Site-To-Site details	N/A	/site-to-site
view system diagnostics	Allows users to view System Diagnostics	Summary	/system
proxy user requests	Allows proxy machines to send requests on the behalf of others	N/A	/proxy
access counters	Allows users to view/modify Counters	Counters	/counters

Component Level Access Policies

Component level access policies govern the following component level authorizations:

Policy	Privilege	Resource Descriptor & Action
view the component	Allows users to view component configuration details	resource="/<component-type>/<component-UUID>" action="R"
modify the component	Allows users to modify component configuration details	resource="/<component-type>/<component-UUID>" action="W"
view provenance	Allows users to view provenance events generated by this component	resource="/provenance-data/<component-type>/<component-UUID>" action="R"
view the data	Allows users to view metadata and content for this component in flowfile queues in outbound connections and through provenance events	resource="/data/<component-type>/<component-UUID>" action="R"
modify the data	Allows users to empty flowfile queues in outbound connections and submit replays through provenance events	resource="/data/<component-type>/<component-UUID>" action="W"

Policy	Privilege	Resource Descriptor & Action
view the policies	Allows users to view the list of users who can view/modify a component	resource="/policies/<component-type>/<component-UUID>" action="R"
modify the policies	Allows users to modify the list of users who can view/modify a component	resource="/policies/<component-type>/<component-UUID>" action="W"
receive data via site-to-site	Allows a port to receive data from NiFi instances	resource="/data-transfer/input-ports/<port-UUID>" action="W"
send data via site-to-site	Allows a port to send data from NiFi instances	resource="/data-transfer/output-ports/<port-UUID>" action="W"

Note: You can apply access policies to all component types except connections. Connection authorizations are inferred by the individual access policies on the source and destination components of the connection, as well as the access policy of the process group containing the components. This is discussed in more detail in the Creating a Connection and Editing a Connection examples below.

Note: In order to access List Queue or Delete Queue for a connection, a user requires permission to the "view the data" and "modify the data" policies on the component. In a clustered environment, all nodes must be be added to these policies as well, as a user request could be replicated through any node in the cluster.

Access Policy Inheritance

An administrator does not need to manually create policies for every component in the dataflow. To reduce the amount of time admins spend on authorization management, policies are inherited from parent resource to child resource. For example, if a user is given access to view and modify a process group, that user can also view and modify the components in the process group. Policy inheritance enables an administrator to assign policies at one time and have the policies apply throughout the entire dataflow.

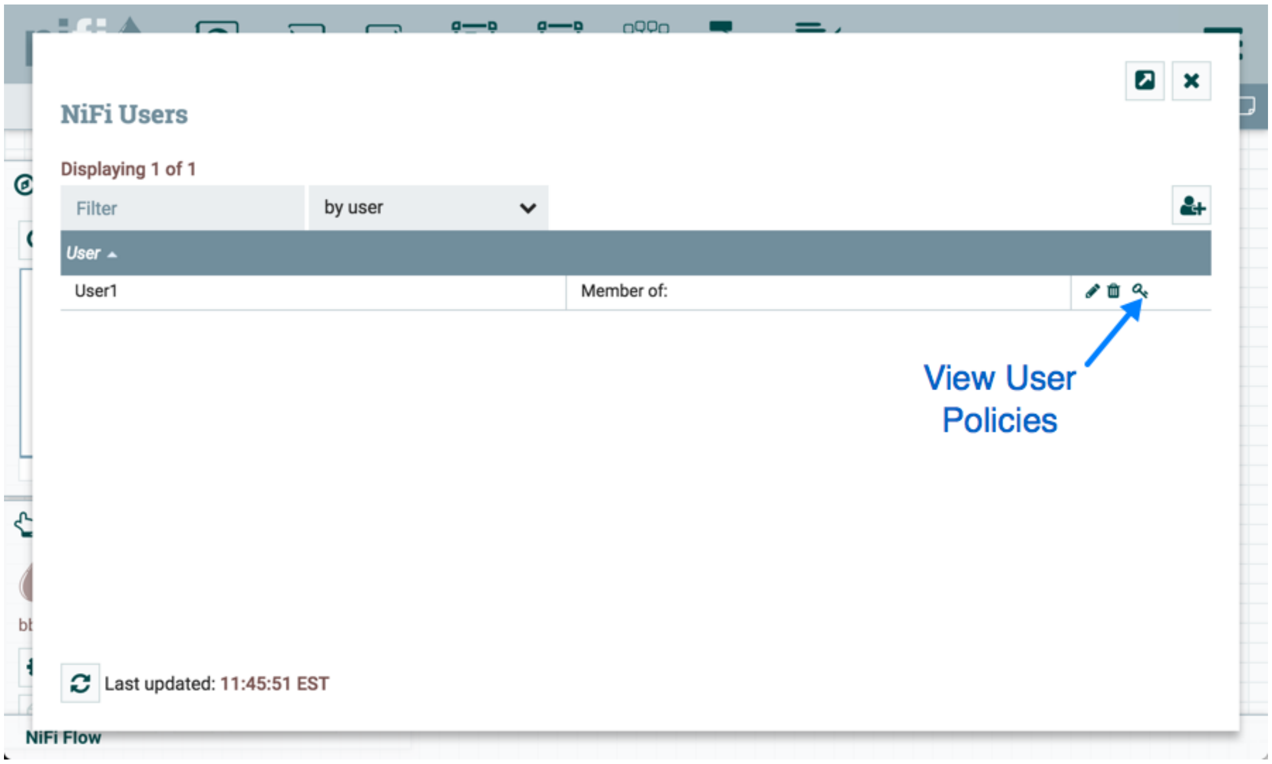
You can override an inherited policy. Overriding a policy removes the inherited policy, breaking the chain of inheritance from parent to child, and creates a replacement policy to add users as desired. Inherited policies and their users can be restored by deleting the replacement policy.

Note: "View the policies" and "modify the policies" component-level access policies are an exception to this inherited behavior. When a user is added to either policy, they are added to the current list of administrators. They do not override higher level administrators. For this reason, only component specific administrators are displayed for the "view the policies" and "modify the policies" access policies.

Note: You cannot modify the users/groups on an inherited policy. Users and groups can only be added or removed from a parent policy or an override policy.

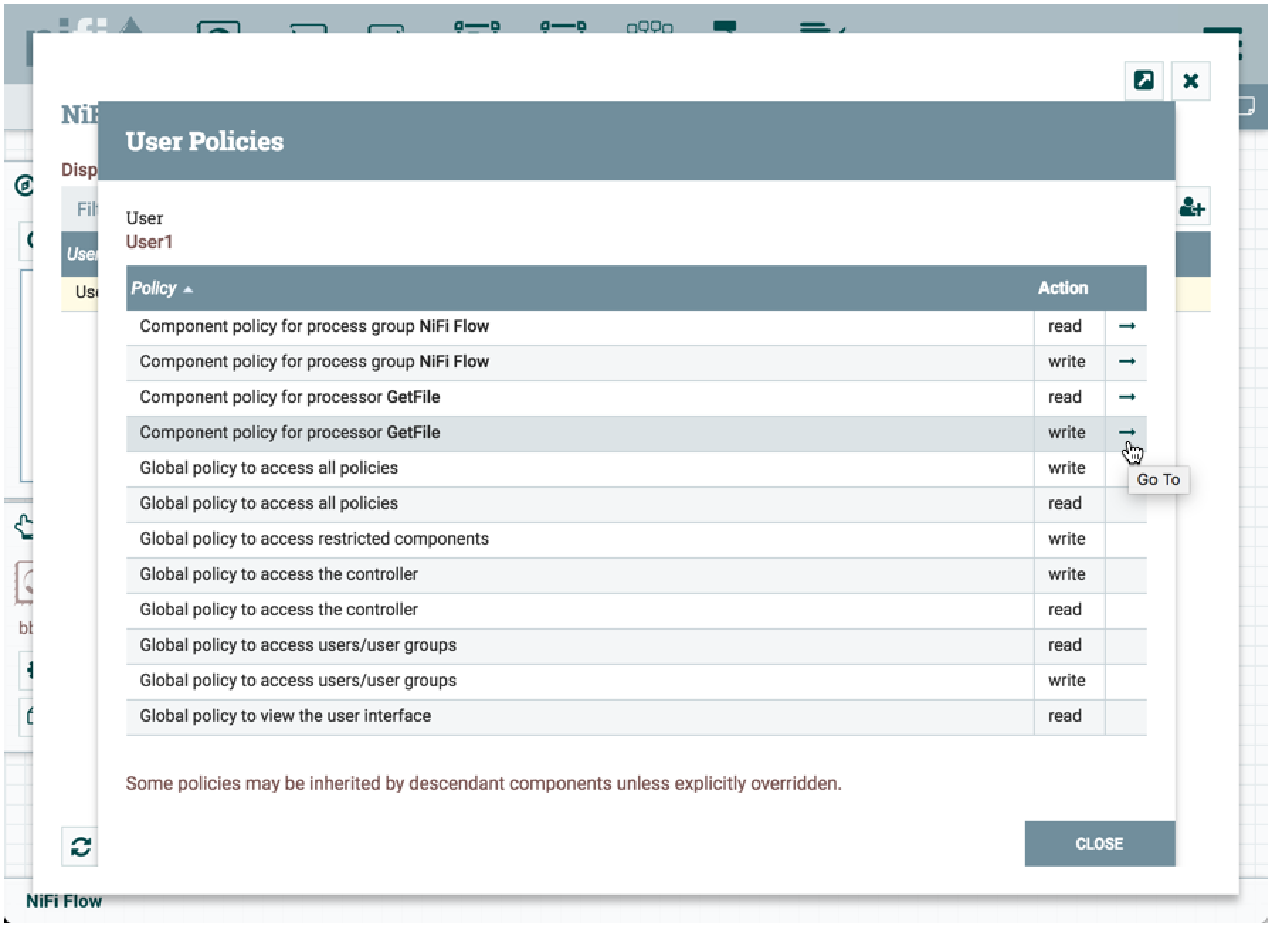
Viewing Policies on Users

From the UI, select "Users" from the Global Menu. This opens the NiFi Users dialog.



Select the View User Policies icon

( ).



Policy	Action
Component policy for process group **NiFi Flow**	read
Component policy for process group **NiFi Flow**	write
Component policy for processor **GetFile**	read
Component policy for processor **GetFile**	write
Global policy to access all policies	write
Global policy to access all policies	read
Global policy to access restricted components	write
Global policy to access the controller	write
Global policy to access the controller	read
Global policy to access users/user groups	read
Global policy to access users/user groups	write
Global policy to view the user interface	read

Some policies may be inherited by descendant components unless explicitly overridden.

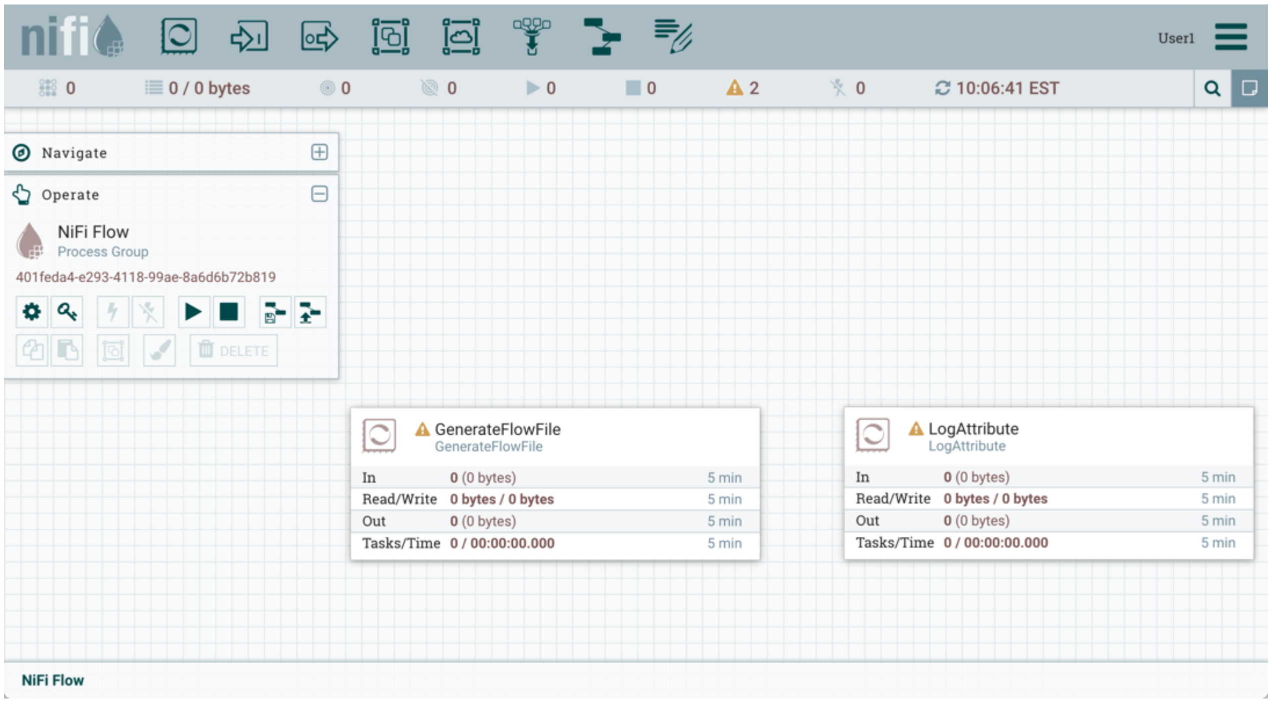
The User Policies window displays the global and component level policies that have been set for the chosen user. Select the Go To icon

( )
to navigate to that component in the canvas.

Access Policy Configuration Examples

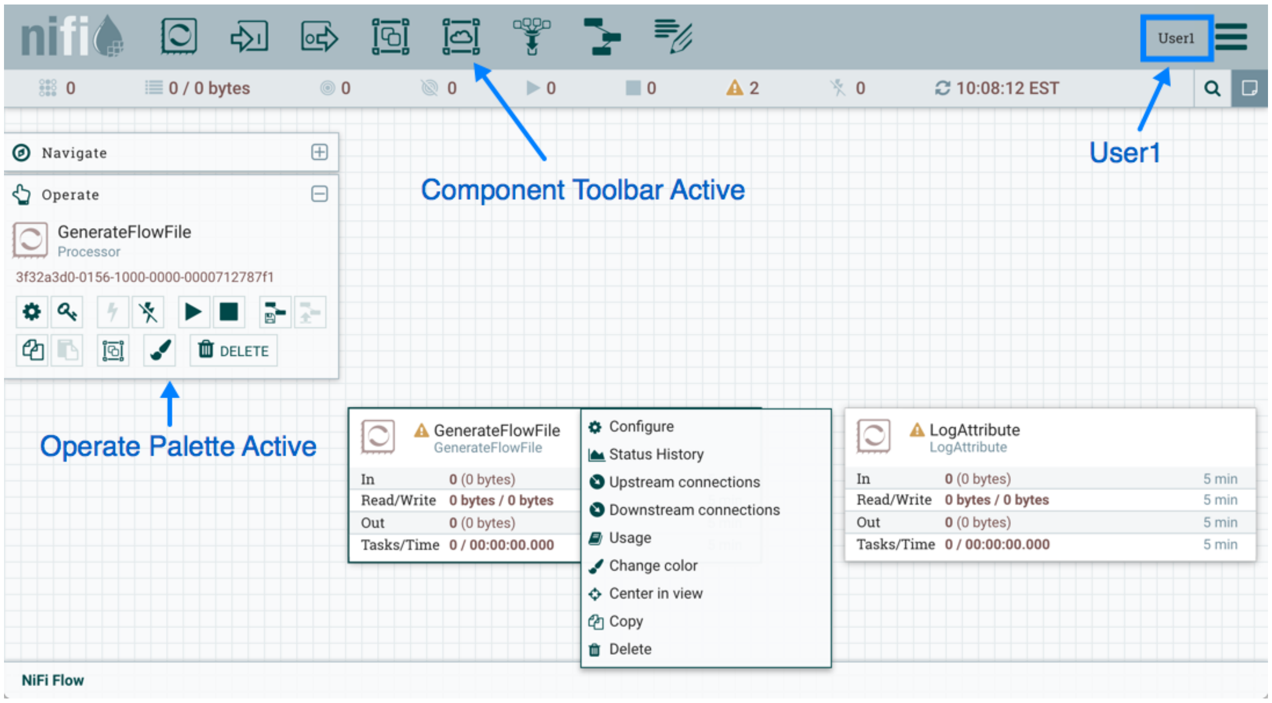
The most effective way to understand how to create and apply access policies is to walk through some common examples. The following scenarios assume User1 is an administrator and User2 is a newly added user that has only been given access to the UI.

Let's begin with two processors on the canvas as our starting point: GenerateFlowFile and LogAttribute.



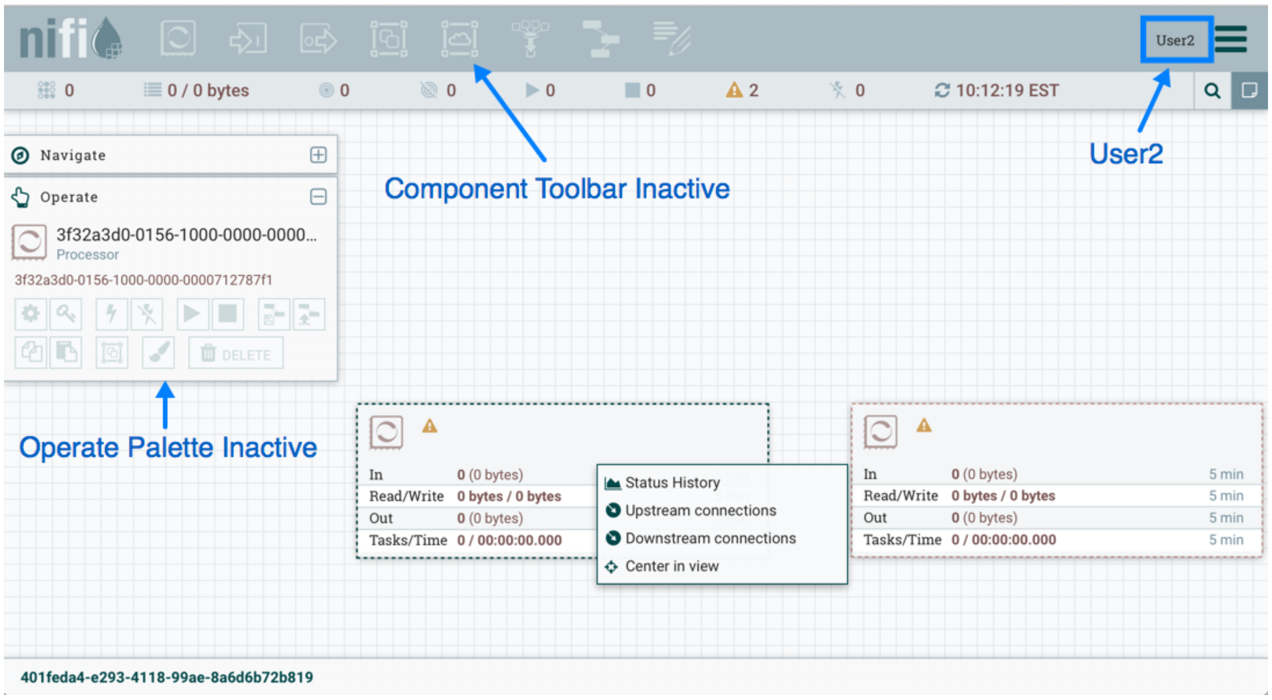
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

User1 can add components to the dataflow and is able to move, edit and connect all processors. The details and properties of the root process group and processors are visible to User1.



User1 wants to maintain their current privileges to the dataflow and its components.

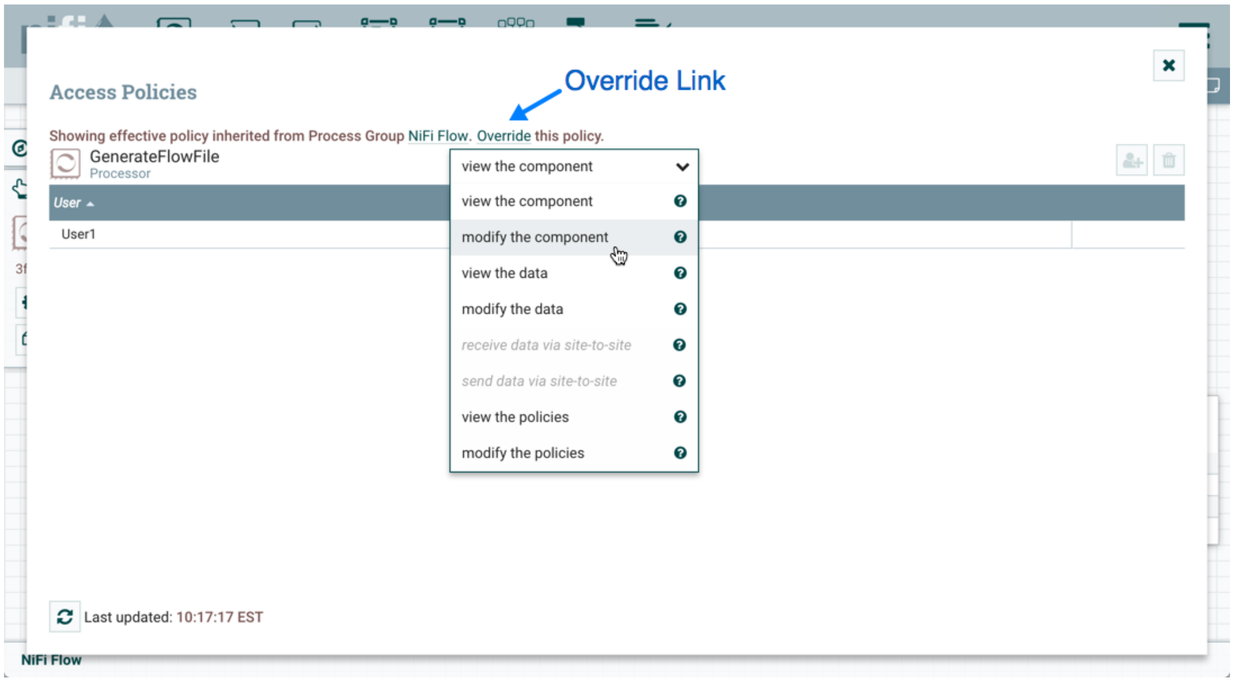
User2 is unable to add components to the dataflow or move, edit, or connect components. The details and properties of the root process group and processors are hidden from User2.



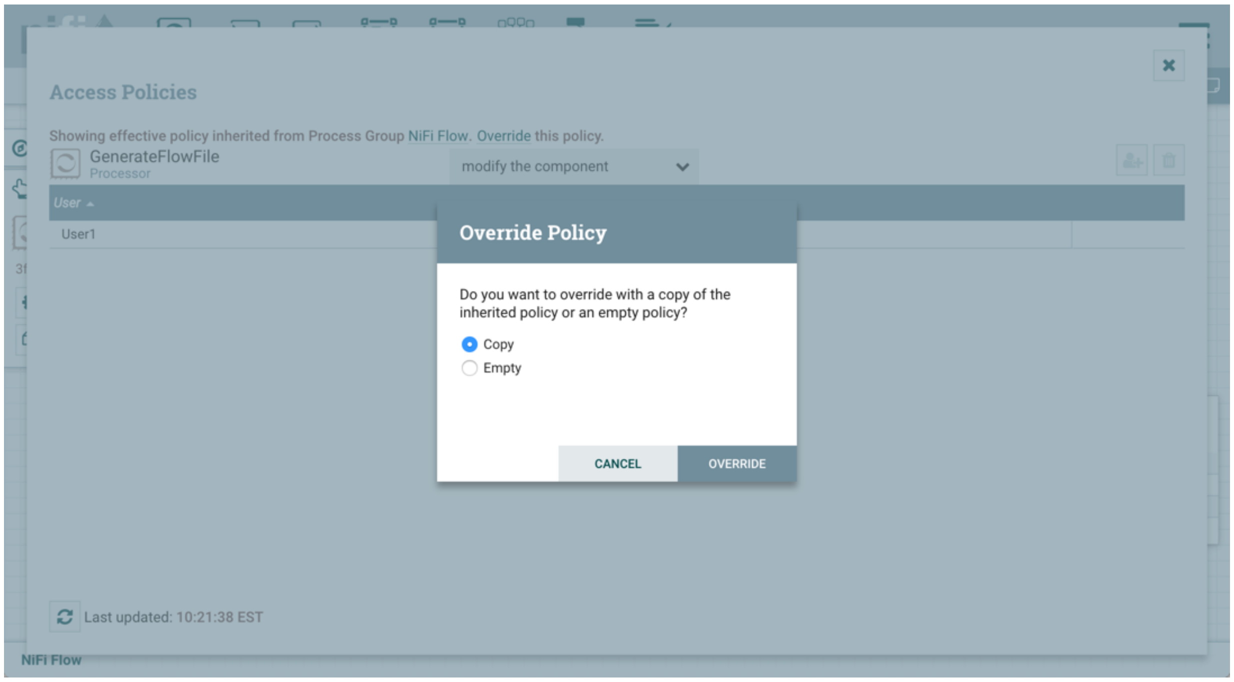
Moving a Processor

To allow User2 to move the GenerateFlowFile processor in the dataflow and only that processor, User1 performs the following steps:

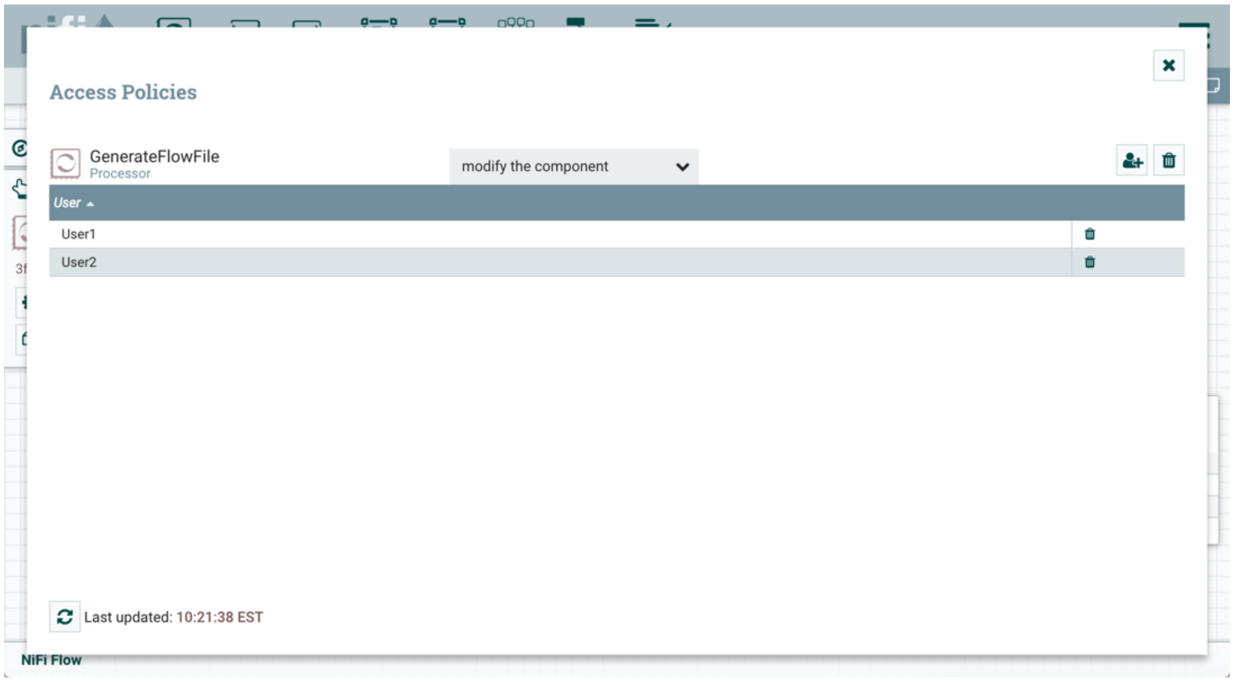
1. Select the GenerateFlowFile processor so that it is highlighted.
2. Select the Access Policies icon ( ) from the Operate palette and the Access Policies dialog opens.
3. Select "modify the component" from the policy drop-down. The "modify the component" policy that currently exists on the processor (child) is the "modify the component" policy inherited from the root process group (parent) on which User1 has privileges.



4. Select the Override link in the policy inheritance message. When creating the replacement policy, you are given a choice to override with a copy of the inherited policy or an empty policy. Select the Override button to create a copy.


Access Policies
Showing effective policy inherited from Process Group NiFi Flow. Override this policy.
GenerateFlowFile
Processor
modify the component
User
User1
Override Policy
Do you want to override with a copy of the inherited policy or an empty policy?
Copy
Empty
CANCEL
OVERRIDE
Last updated: 10:21:38 EST
NiFi Flow

5. On the replacement policy that is created, select the Add User icon ( ). Find or enter User2 in the User Identity field and select OK. With these changes, User1 maintains the ability to move both processors on the canvas. User2 can now move the GenerateFlowFile processor but cannot move the LogAttribute processor.



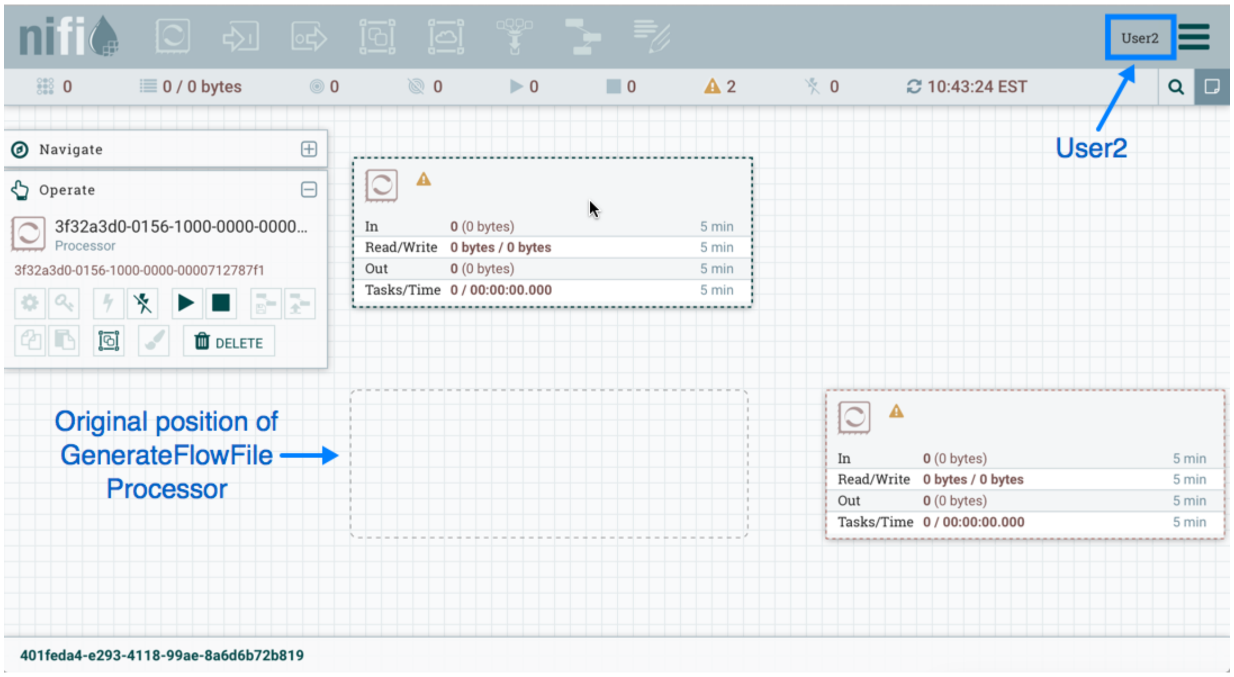
Access Policies

GenerateFlowFile
Processor

modify the component

User
User1
User2

Last updated: 10:21:38 EST


User2
0
0 / 0 bytes
0
0
0
0
2
0
10:43:24 EST
Navigate
Operate
3f32a3d0-0156-1000-0000-0000...
Processor
3f32a3d0-0156-1000-0000-0000712787f1
DELETE
Original position of GenerateFlowFile Processor
401feda4-e293-4118-99ae-8a6d6b72b819

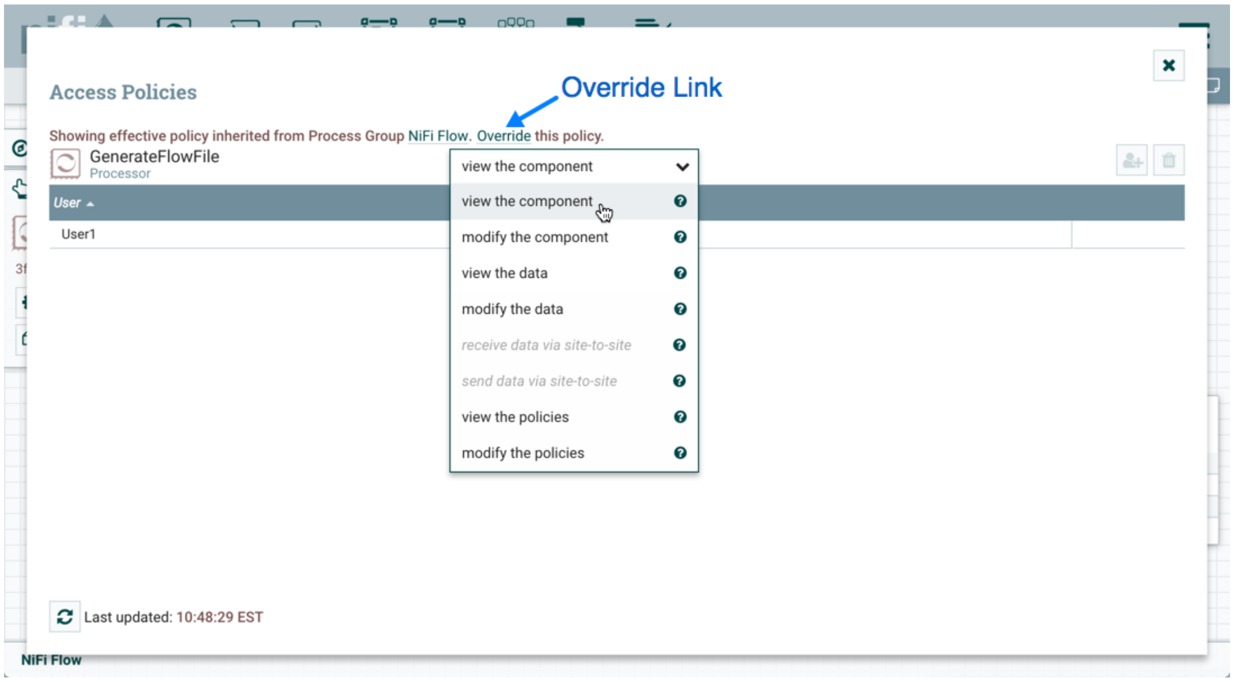
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Editing a Processor

In the "Moving a Processor" example above, User2 was added to the "modify the component" policy for GenerateFlowFile. Without the ability to view the processor properties, User2 is unable to modify the processor's configuration. In order to edit a component, a user must be on both the "view the component" and "modify the component" policies. To implement this, User1 performs the following steps:

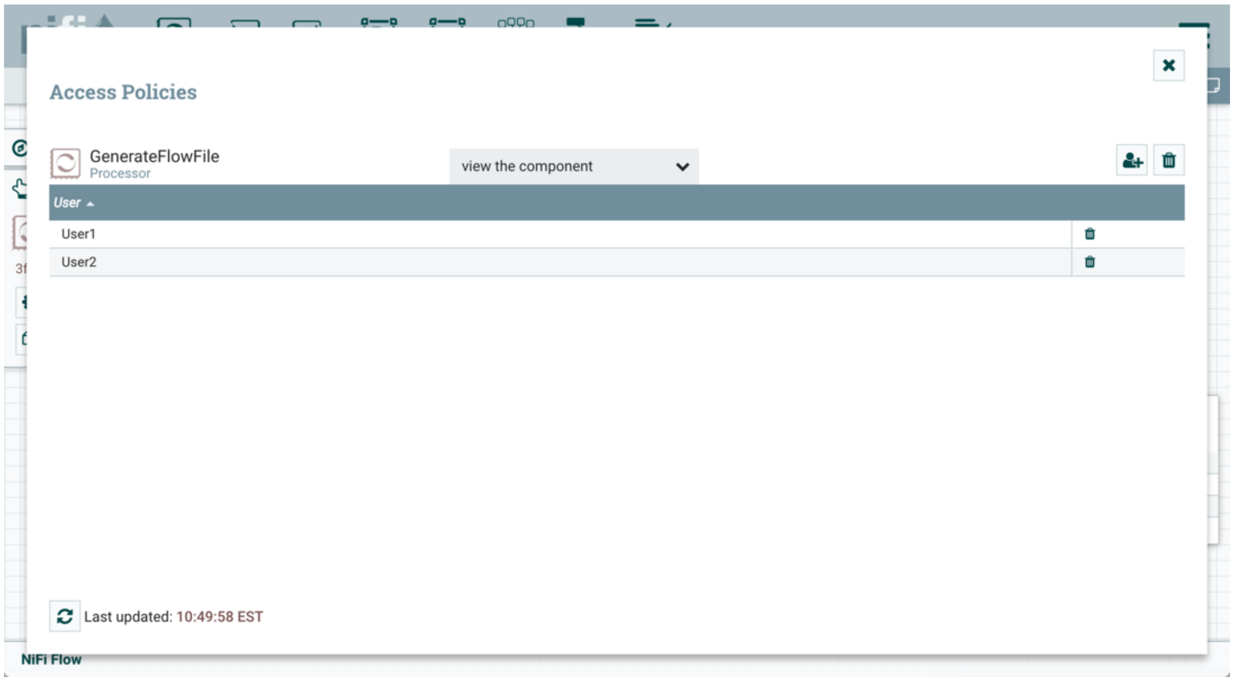
1. Select the GenerateFlowFile processor.
2. Select the Access Policies icon ( ) from the Operate palette and the Access Policies dialog opens.

3. Select "view the component" from the policy drop-down. The view the component" policy that currently exists on the processor (child) is the "view the component" policy inherited from the root process group (parent) on which User1 has privileges.



4. Select the Override link in the policy inheritance message, keep the default of Copy policy and select the Override button.

5. On the override policy that is created, select the Add User icon ( ). Find or enter User2 in the User Identity field and select OK. With these changes, User1 maintains the ability to view and edit the processors on the canvas. User2 can now view and edit the GenerateFlowFile processor.

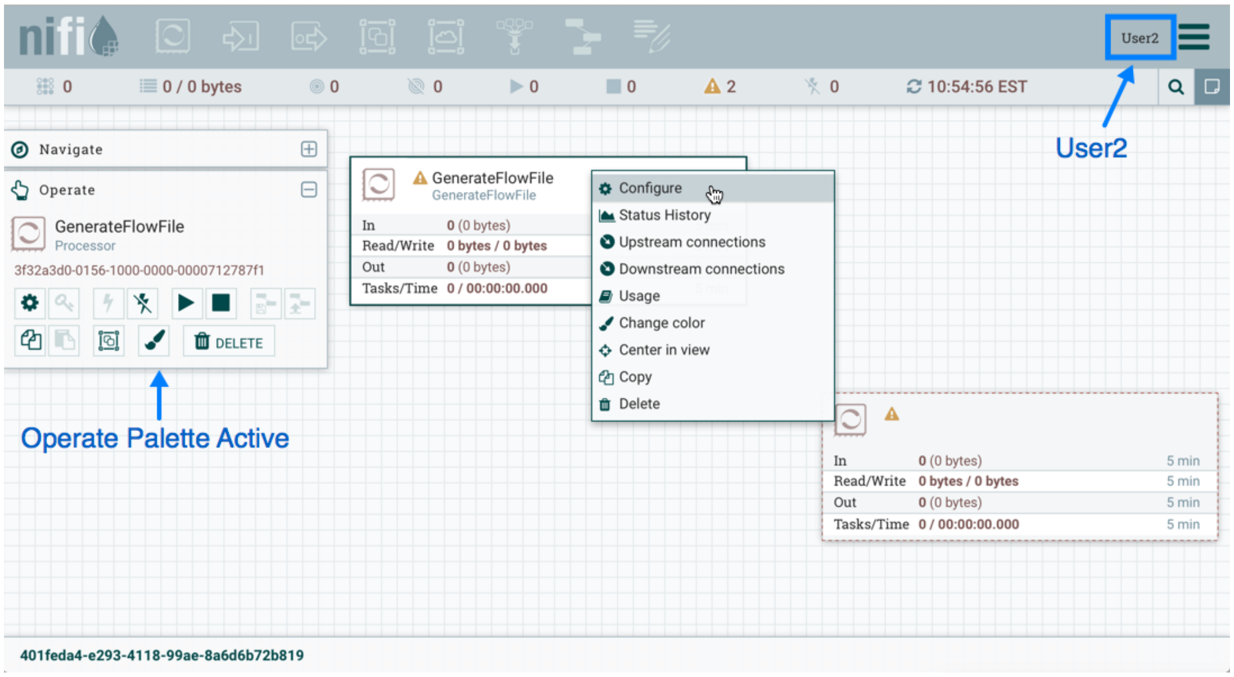

Access Policies

GenerateFlowFile
Processor

view the component

User
User1
User2

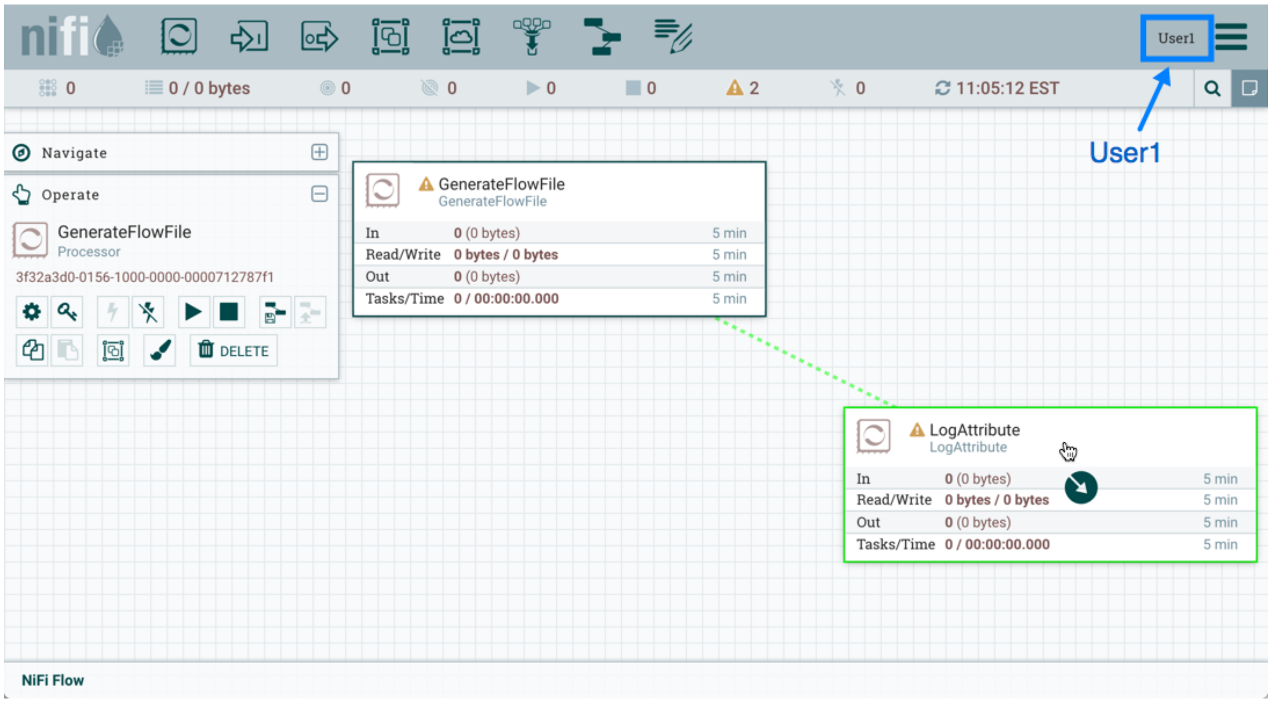
Last updated: 10:49:58 EST



In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Creating a Connection

With the access policies configured as discussed in the previous two examples, User1 is able to connect GenerateFlowFile to LogAttribute:



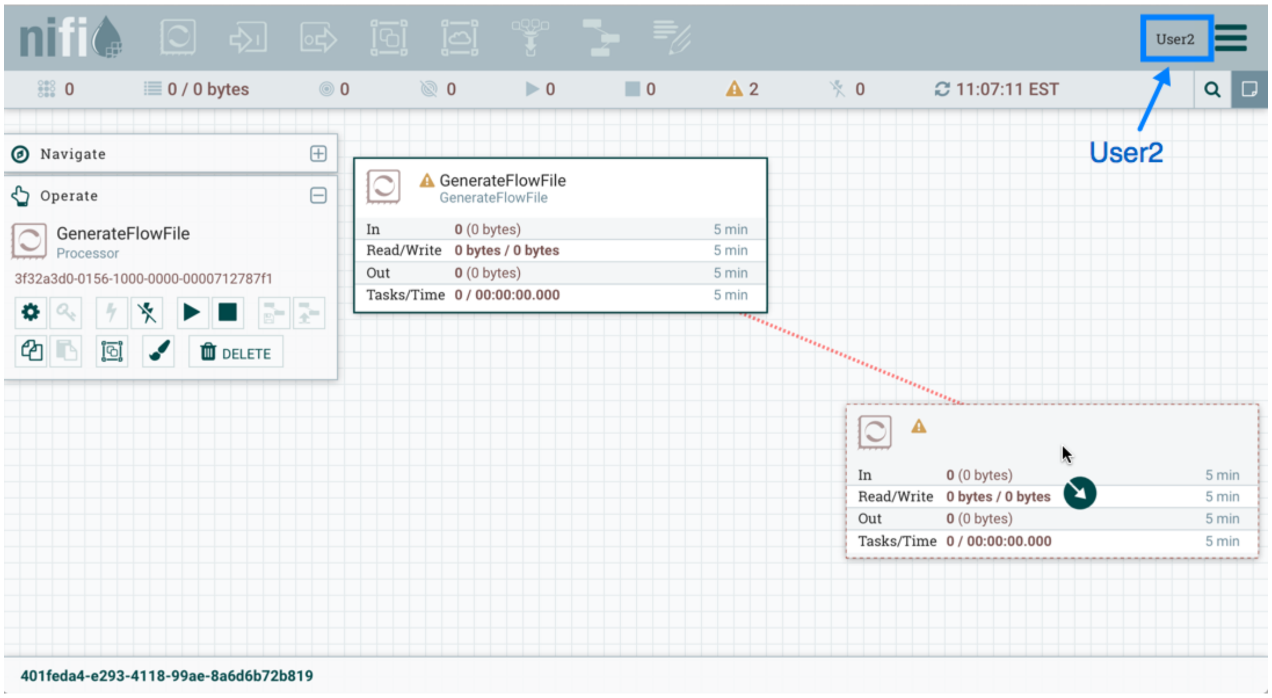
GenerateFlowFile
GenerateFlowFile

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

LogAttribute
LogAttribute

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

User2 cannot make the connection:



GenerateFlowFile
GenerateFlowFile

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

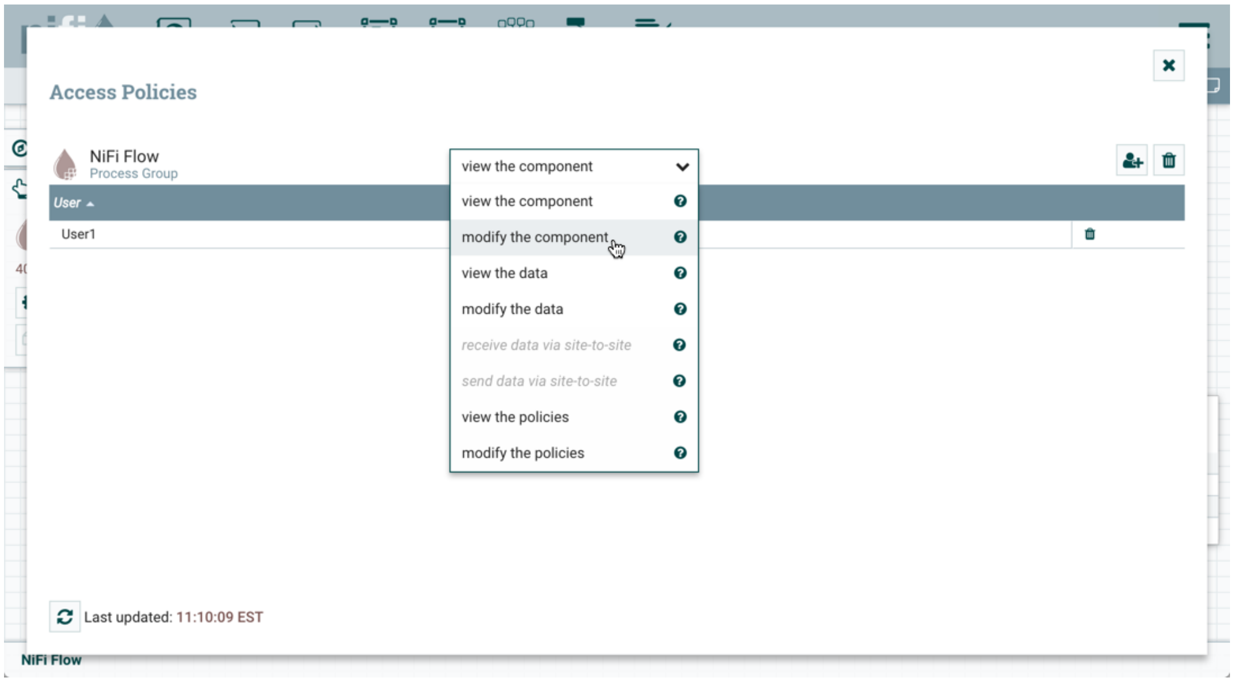
This is because:

- User2 does not have modify access on the process group.
- Even though User2 has view and modify access to the source component (GenerateFlowFile), User2 does not have an access policy on the destination component (LogAttribute).

To allow User2 to connect GenerateFlowFile to LogAttribute, as User1:

1. Select the root process group. The Operate palette is updated with details for the root process group.

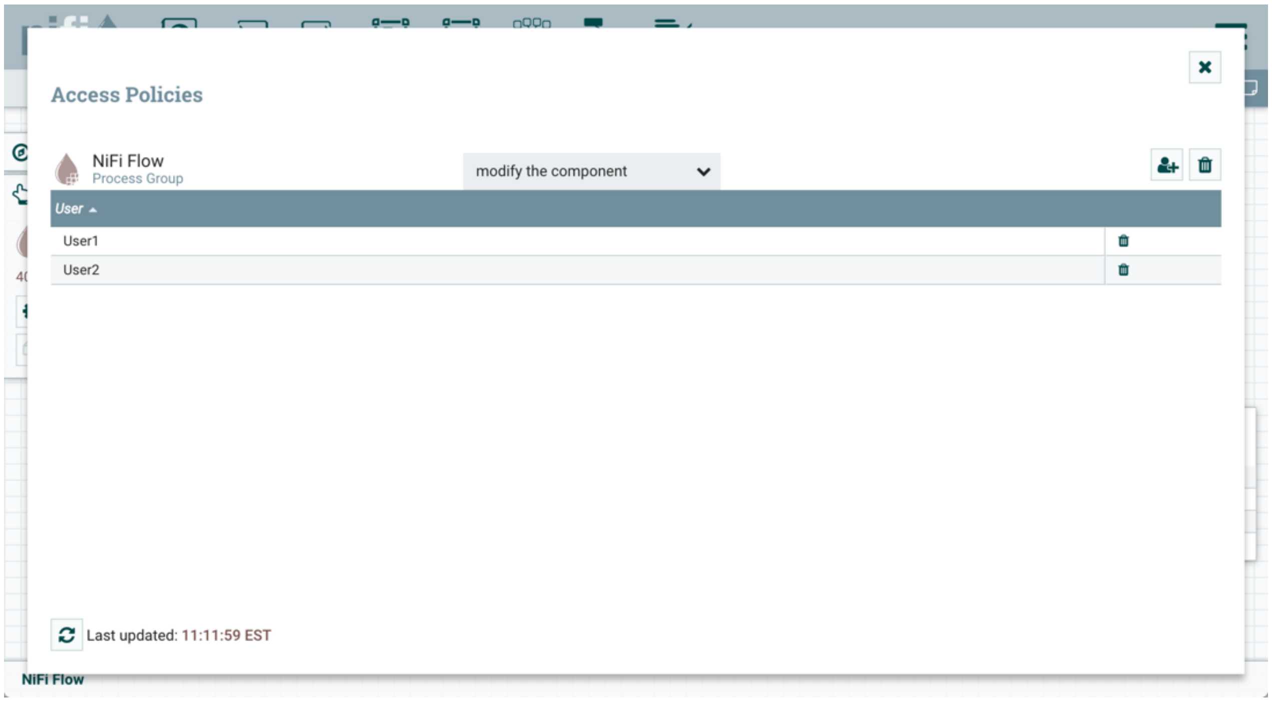
2. Select the Access Policies icon ( ) from the Operate palette and the Access Policies dialog opens.
3. Select "modify the component" from the policy drop-down.



Access Policies

NiFi Flow
Process Group

4. Select the Add User icon ( ). Find or enter User2 and select OK.

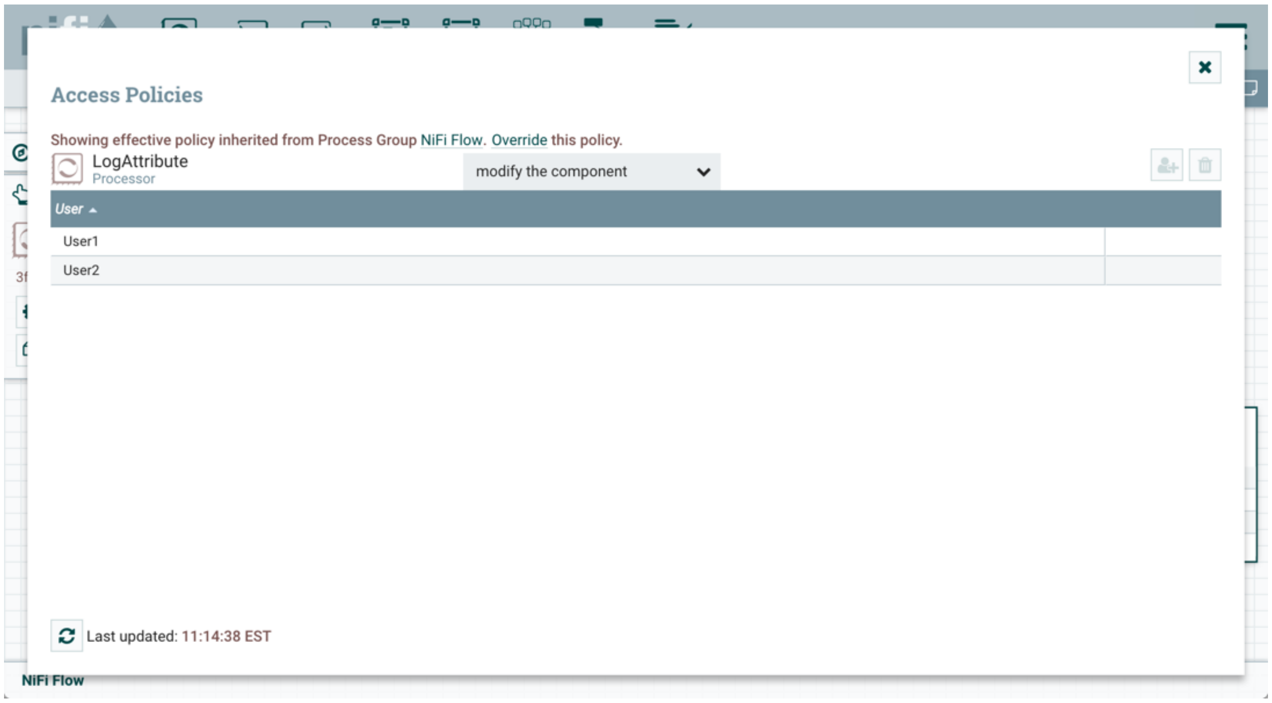


Access Policies

NiFi Flow
Process Group

User
User1
User2

By adding User2 to the "modify the component" policy on the process group, User2 is added to the "modify the component" policy on the LogAttribute processor by policy inheritance. To confirm this, highlight the LogAttribute processor and select the Access Policies icon ( ) from the Operate palette:


Access Policies

Showing effective policy inherited from Process Group NiFi Flow. Override this policy.

LogAttribute
Processor

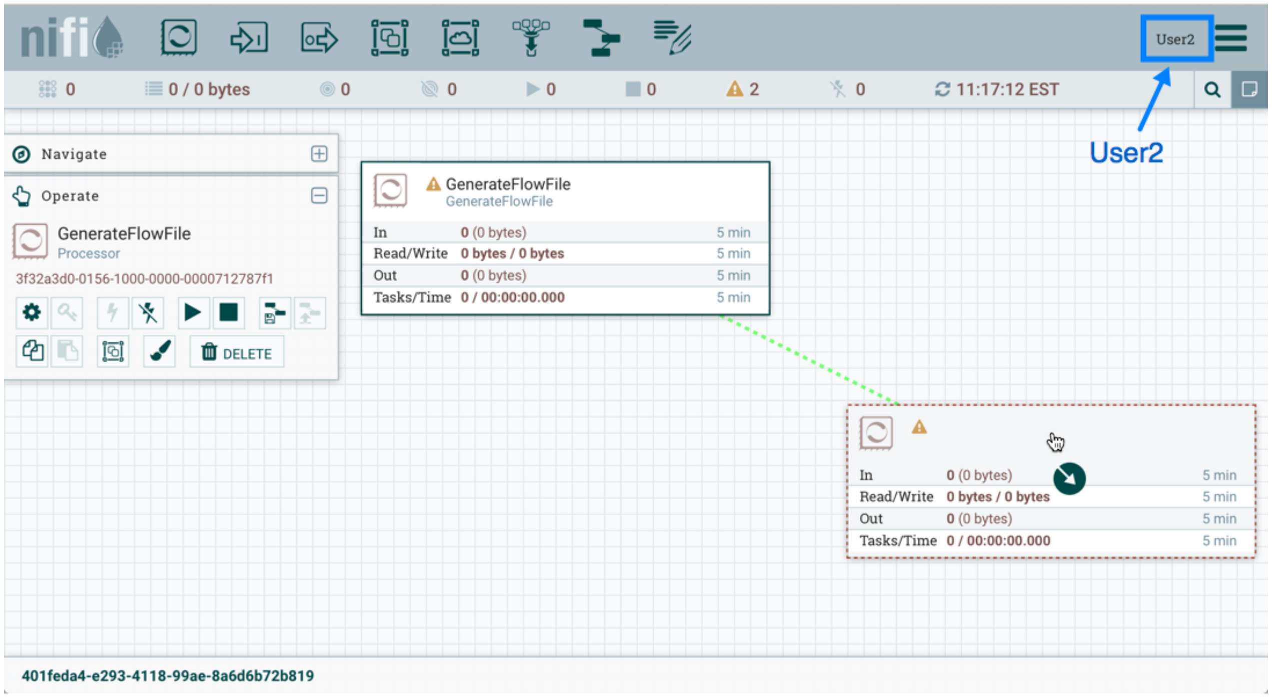
modify the component

User
User1
User2

Last updated: 11:14:38 EST

NiFi Flow

With these changes, User2 can now connect the GenerateFlowFile processor to the LogAttribute processor.

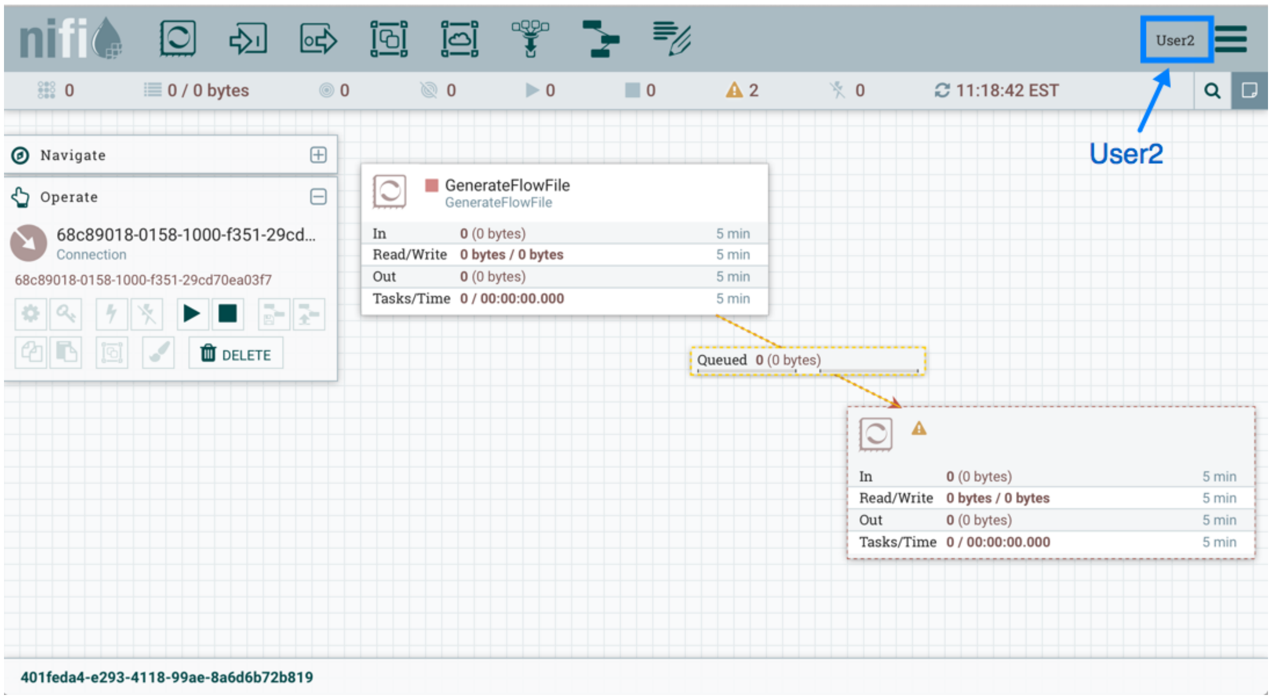


GenerateFlowFile
GenerateFlowFile

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

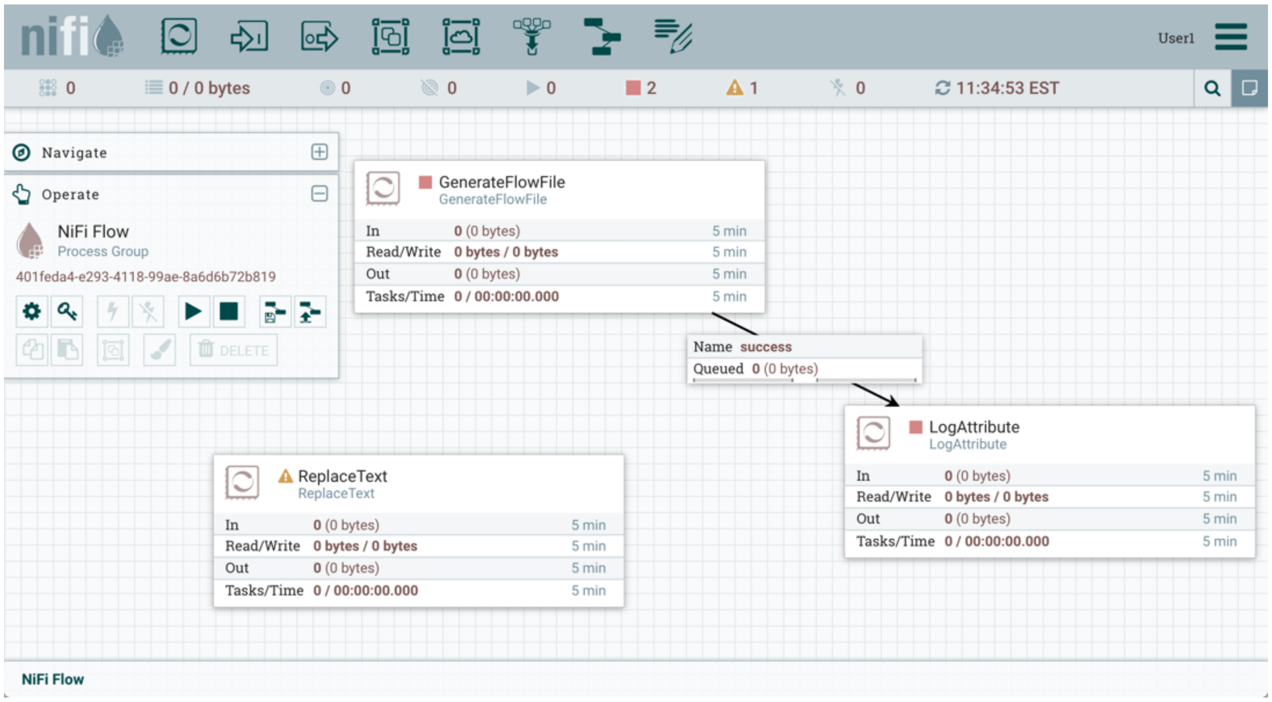
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

401feda4-e293-4118-99ae-8a6d6b72b819

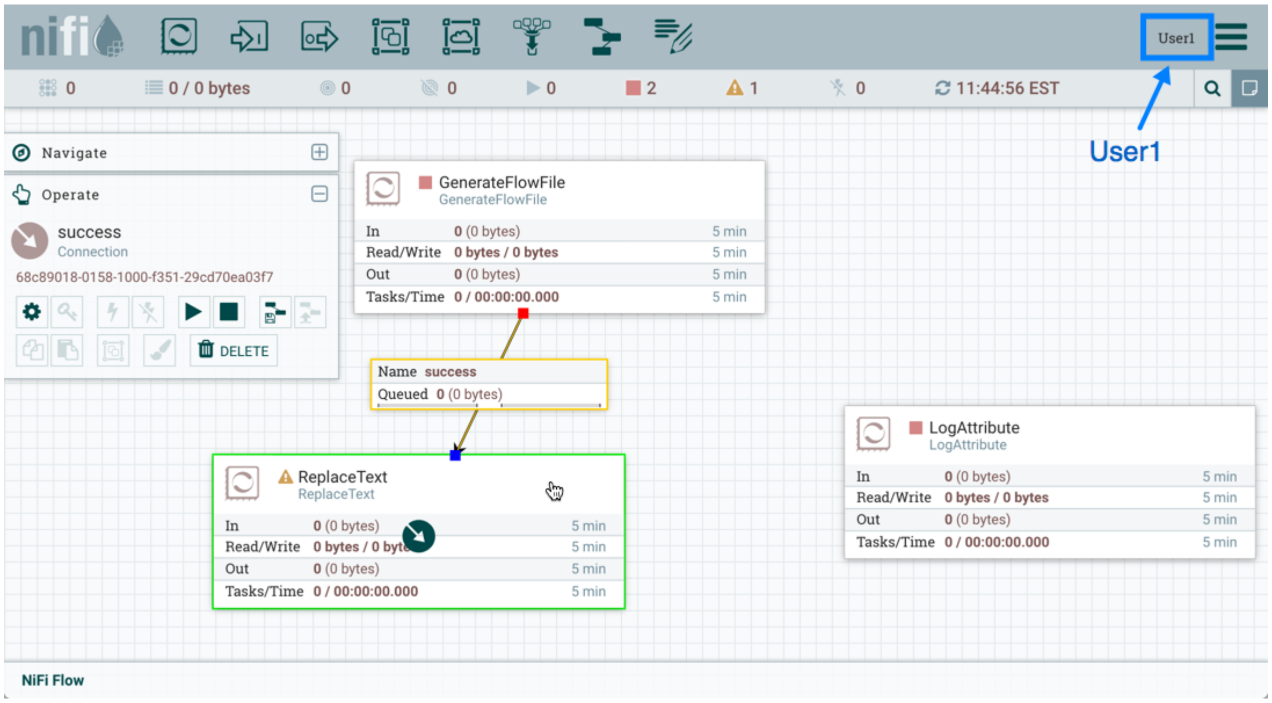

User2
User2
Navigate
Operate
68c89018-0158-1000-f351-29cd...
Connection
68c89018-0158-1000-f351-29cd70ea03f7
DELETE
GenerateFlowFile
Queued 0 (0 bytes)
401feda4-e293-4118-99ae-8a6d6b72b819

Editing a Connection

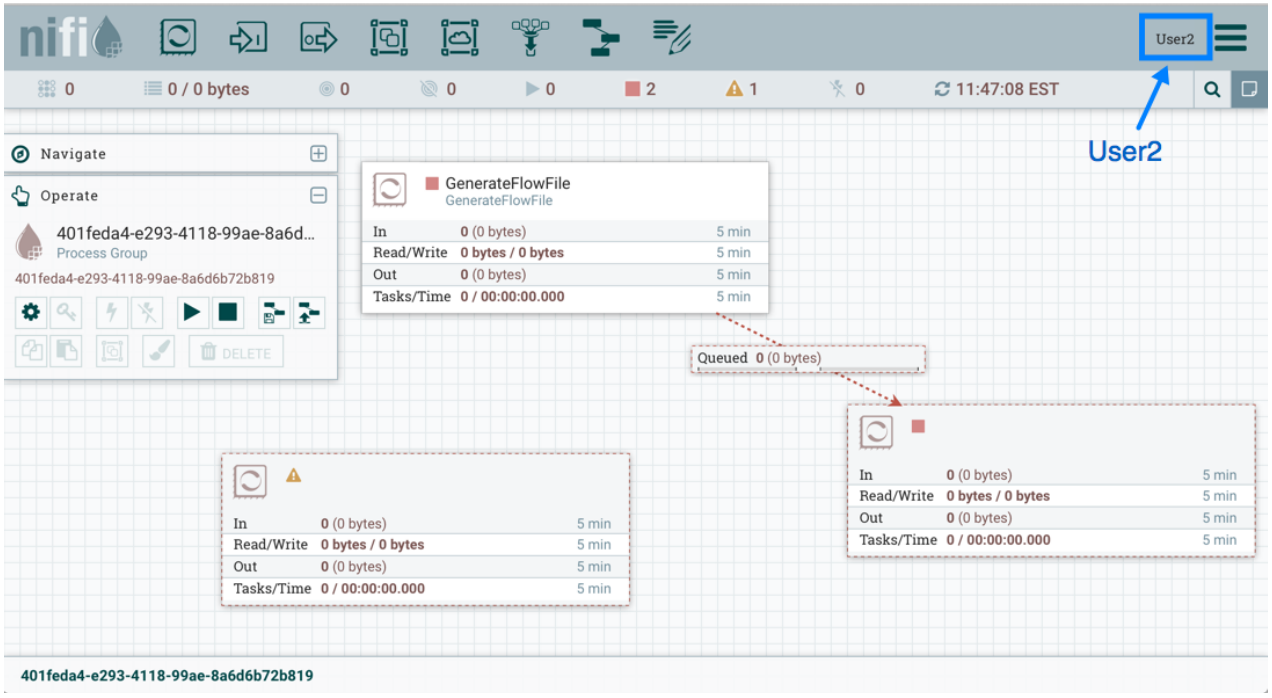
Assume User1 or User2 adds a ReplaceText processor to the root process group:



User1 can select and change the existing connection (between GenerateFlowFile to LogAttribute) to now connect GenerateFlowFile to ReplaceText:



User 2 is unable to perform this action.

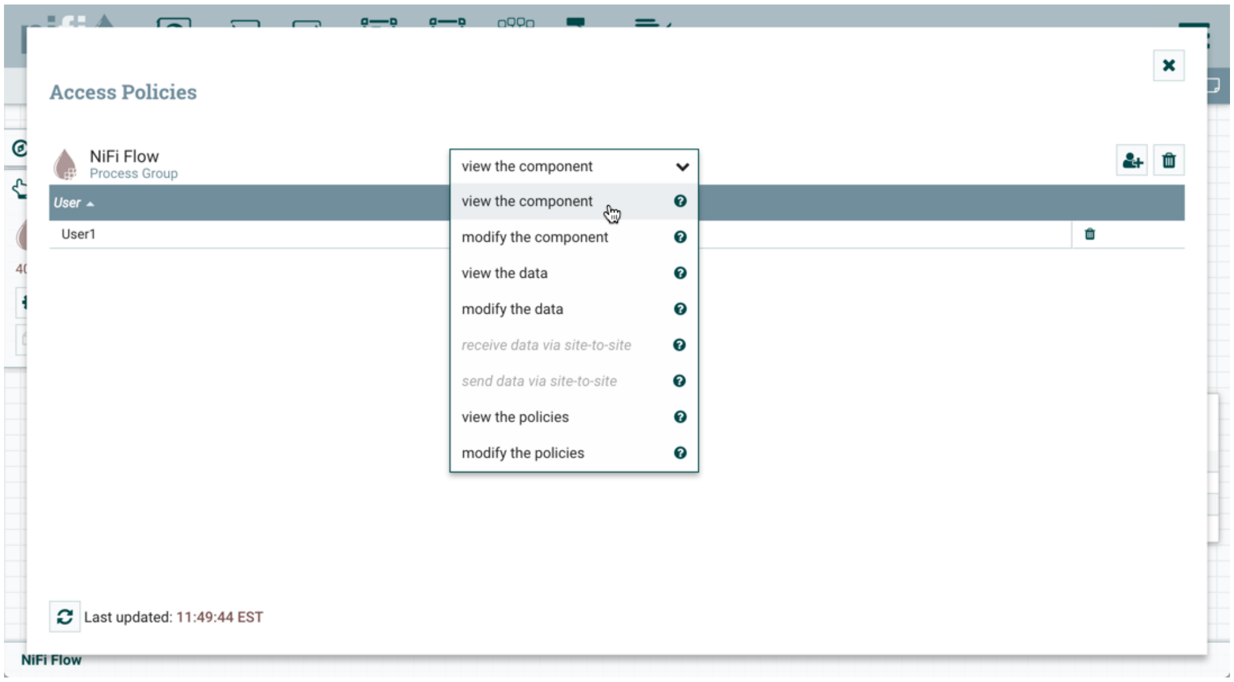

User2
0
0 / 0 bytes
0
0
0
2
1
0
11:47:08 EST
User2
Navigate
Operate
401feda4-e293-4118-99ae-8a6d...
Process Group
401feda4-e293-4118-99ae-8a6d6b72b819
DELETE
GenerateFlowFile
GenerateFlowFile
Queued 0 (0 bytes)
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
401feda4-e293-4118-99ae-8a6d6b72b819

To allow User2 to connect GenerateFlowFile to ReplaceText, as User1:

1. Select the root process group. The Operate palette is updated with details for the root process group.
2. Select the Access Policies icon

( ).

3. Select "view the component" from the policy drop-down.



Access Policies

NiFi Flow
Process Group

view the component

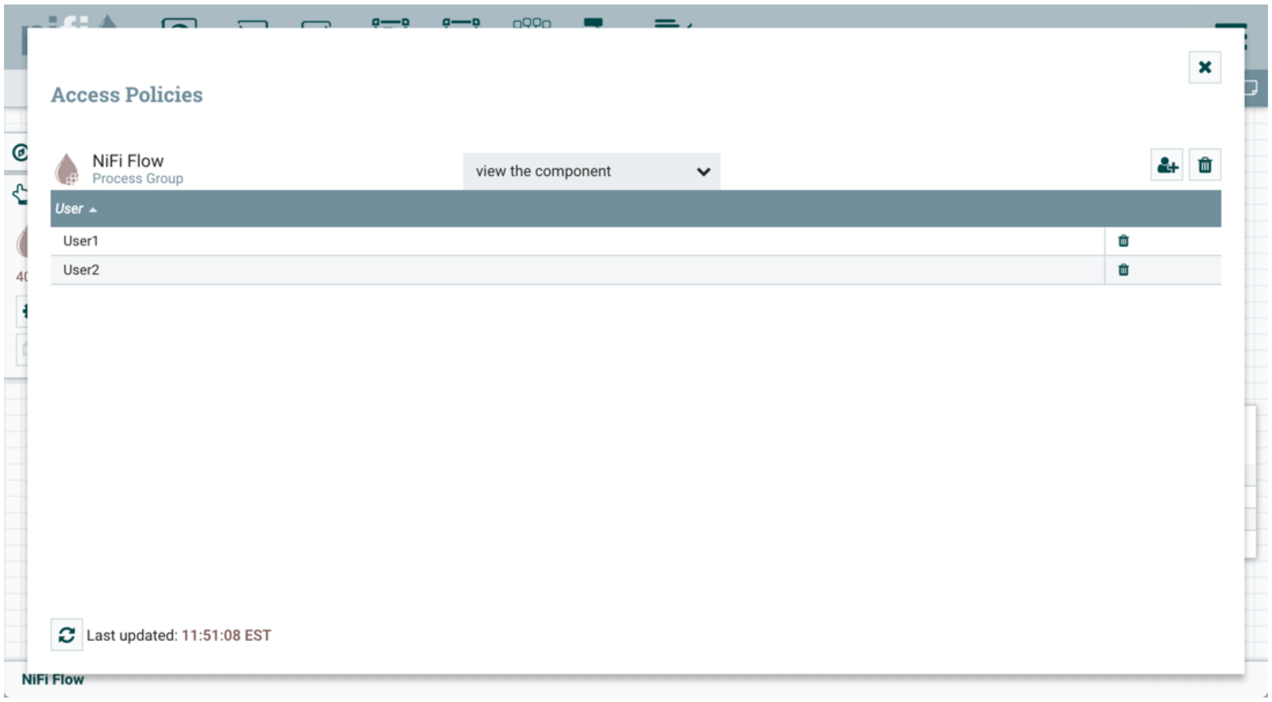
User

User1

Last updated: 11:49:44 EST

NiFi Flow

4. Select the Add User icon ( ). Find or enter User2 and select OK.



Access Policies

NiFi Flow
Process Group

view the component

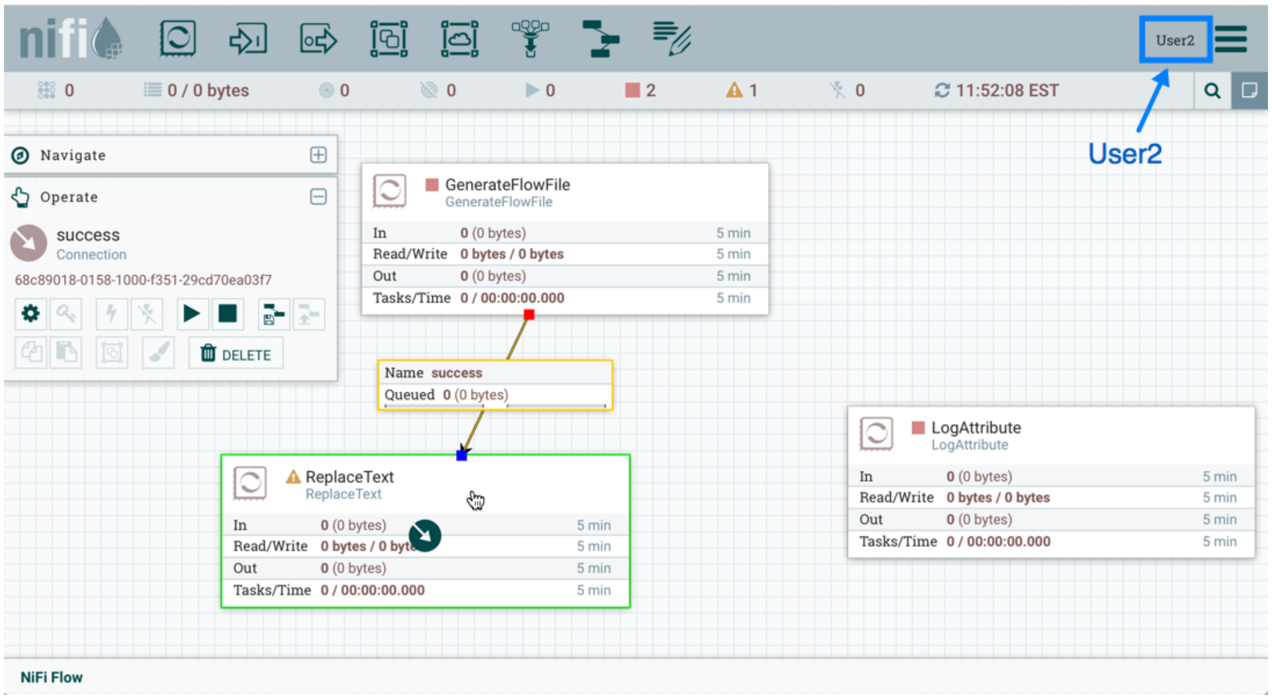
User

User1
User2

Last updated: 11:51:08 EST

NiFi Flow

Being added to both the view and modify policies for the process group, User2 can now connect the GenerateFlowFile processor to the ReplaceText processor.



In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Encryption Configuration

This section provides an overview of the capabilities of NiFi to encrypt and decrypt data.

The EncryptContent processor allows for the encryption and decryption of data, both internal to NiFi and integrated with external systems, such as openssl and other data sources and consumers.

Key Derivation Functions

Key Derivation Functions (KDF) are mechanisms by which human-readable information, usually a password or other secret information, is translated into a cryptographic key suitable for data protection. Currently, KDFs are ingested by CipherProvider implementations and return a fully-initialized Cipher object to be used for encryption or decryption. Due to the use of a CipherProviderFactory, the KDFs are not customizable at this time. Future enhancements will include the ability to provide custom cost parameters to the KDF at initialization time. As a work-around, CipherProvider instances can be initialized with custom cost parameters in the constructor but this is not currently supported by the CipherProviderFactory. Here are the KDFs currently supported by NiFi (primarily in the EncryptContent processor for password-based encryption (PBE)) and relevant notes:

- NiFi Legacy KDF
  - The original KDF used by NiFi for internal key derivation for PBE, this is 1000 iterations of the MD5 digest over the concatenation of the password and 8 or 16 bytes of random salt (the salt length depends on the selected cipher block size).
  - This KDF is deprecated as of NiFi 0.5.0 and should only be used for backwards compatibility to decrypt data that was previously encrypted by a legacy version of NiFi.
- OpenSSL PKCS#5 v1.5 EVP_BytesToKey
  - This KDF was added in v0.4.0.
  - This KDF is provided for compatibility with data encrypted using OpenSSL's default PBE, known as EVP_BytesToKey. This is a single iteration of MD5 over the concatenation of the password and 8 bytes of random ASCII salt. OpenSSL recommends using PBKDF2 for key derivation but does not expose the

library method necessary to the command-line tool, so this KDF is still the de facto default for command-line encryption.

- Bcrypt
  - This KDF was added in v0.5.0.
  - https://en.wikipedia.org/wiki/Bcrypt is an adaptive function based on the https://en.wikipedia.org/wiki/Blowfish_(cipher) cipher. This KDF is strongly recommended as it automatically incorporates a random 16 byte salt, configurable cost parameter (or "work factor"), and is hardened against brute-force attacks using https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units (which share memory between cores) by requiring access to "large" blocks of memory during the key derivation. It is less resistant to https://en.wikipedia.org/wiki/Field-programmable_gate_array brute-force attacks where the gate arrays have access to individual embedded RAM blocks.
  - Because the length of a Bcrypt-derived key is always 184 bits, the complete output is then fed to a SHA-512 digest and truncated to the desired key length. This provides the benefit of the avalanche effect on the formatted input.
  - The recommended minimum work factor is 12 (212 key derivation rounds) (as of 2/1/2016 on commodity hardware) and should be increased to the threshold at which legitimate systems will encounter detrimental delays (see schedule below or use BcryptCipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongWorkFactor() to calculate safe minimums).
  - The salt format is $2a$10$ABCDEFGHIJKLMNOPQRSTUV. The salt is delimited by $ and the three sections are as follows:
    - 2a - the version of the format. An extensive explanation can be found http://blog.ircmaxell.com/2012/12/seven-ways-to-screw-up-bcrypt.html. NiFi currently uses 2a for all salts generated internally.
    - 10 - the work factor. This is actually the log2 value, so the total iteration count would be 210 in this case.
    - ABCDEFGHIJKLMNOPQRSTUV - the 22 character, Base64-encoded, unpadded, raw salt value. This decodes to a 16 byte salt used in the key derivation.
- Scrypt
  - This KDF was added in v0.5.0.
  - https://en.wikipedia.org/wiki/Scrypt is an adaptive function designed in response to bcrypt. This KDF is recommended as it requires relatively large amounts of memory for each derivation, making it resistant to hardware brute-force attacks.
  - The recommended minimum cost is N=214, r=8, p=1 (as of 2/1/2016 on commodity hardware) and should be increased to the threshold at which legitimate systems will encounter detrimental delays (see schedule below or use ScryptCipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongParameters() to calculate safe minimums).
  - The salt format is $s0$e0101$ABCDEFGHIJKLMNOPQRSTUV. The salt is delimited by $ and the three sections are as follows:
    - s0 - the version of the format. NiFi currently uses s0 for all salts generated internally.
    - e0101 - the cost parameters. This is actually a hexadecimal encoding of N, r, p using shifts. This can be formed/parsed using Scrypt#encodeParams() and Scrypt#parseParameters().
      - Some external libraries encode N, r, and p separately in the form $400$1$1$. A utility method is available at ScryptCipherProvider#translateSalt() which will convert the external form to the internal form.
    - ABCDEFGHIJKLMNOPQRSTUV - the 12-44 character, Base64-encoded, unpadded, raw salt value. This decodes to a 8-32 byte salt used in the key derivation.
- PBKDF2
  - This KDF was added in v0.5.0.
  - https://en.wikipedia.org/wiki/PBKDF2 is an adaptive derivation function which uses an internal pseudorandom function (PRF) and iterates it many times over a password and salt (at least 16 bytes).
  - The PRF is recommended to be HMAC/SHA-256 or HMAC/SHA-512. The use of an HMAC cryptographic hash function mitigates a length extension attack.

    - The recommended minimum number of iterations is 160,000 (as of 2/1/2016 on commodity hardware). This number should be doubled every two years (see schedule below or use PBKDF2CipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongIterationCount() to calculate safe minimums).
    - This KDF is not memory-hard (can be parallelized massively with commodity hardware) but is still recommended as sufficient by http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf and many cryptographers (when used with a proper iteration count and HMAC cryptographic hash function).
- None
    - This KDF was added in v0.5.0.
    - This KDF performs no operation on the input and is a marker to indicate the raw key is provided to the cipher. The key must be provided in hexadecimal encoding and be of a valid length for the associated cipher/algorithm.

Additional Resources

- http://stackoverflow.com/a/30308723/70465
- http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf
- https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Work_Factor
- http://security.stackexchange.com/a/3993/16485
- http://blog.ircmaxell.com/2014/03/why-i-dont-recommend-scrypt.html
- http://security.stackexchange.com/a/26253/16485
- http://security.stackexchange.com/a/6415/16485
- http://wildlyinaccurate.com/bcrypt-choosing-a-work-factor/
- https://www.openssl.org/docs/man1.1.0/crypto/EVP_BytesToKey.html
- http://security.stackexchange.com/a/29139/16485

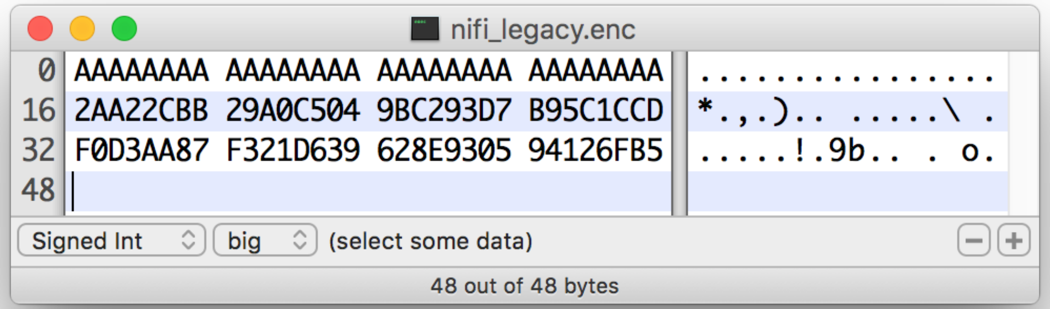
Salt and IV Encoding

Initially, the EncryptContent processor had a single method of deriving the encryption key from a user-provided password. This is now referred to as NiFiLegacy mode, effectively MD5 digest, 1000 iterations. In v0.4.0, another method of deriving the key, OpenSSL PKCS#5 v1.5 EVP_BytesToKey was added for compatibility with content encrypted outside of NiFi using the openssl command-line tool. Both of these Key Derivation Functions (KDF) had hard-coded digest functions and iteration counts, and the salt format was also hard-coded. With v0.5.0, additional KDFs are introduced with variable iteration counts, work factors, and salt formats. In addition, raw keyed encryption was also introduced. This required the capacity to encode arbitrary salts and Initialization Vectors (IV) into the cipher stream in order to be recovered by NiFi or a follow-on system to decrypt these messages.

For the existing KDFs, the salt format has not changed.

NiFi Legacy

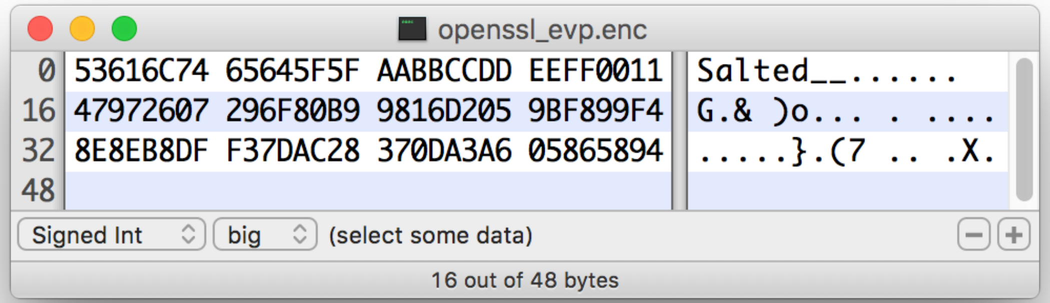
The first 8 or 16 bytes of the input are the salt. The salt length is determined based on the selected algorithm's cipher block length. If the cipher block size cannot be determined (such as with a stream cipher like RC4), the default value of 8 bytes is used. On decryption, the salt is read in and combined with the password to derive the encryption key and IV.



```
nifi_legacy.enc
 0 AAAAAAAA AAAAAAAA AAAAAAAA AAAAAAAA  ................
16 2AA22CBB 29A0C504 9BC293D7 B95C1CCD  *.,.).. .....\ .
32 F0D3AA87 F321D639 628E9305 94126FB5  .....!.9b.. . o.
48
Signed Int   big   (select some data)
48 out of 48 bytes
```

OpenSSL PKCS#5 v1.5 EVP_BytesToKey

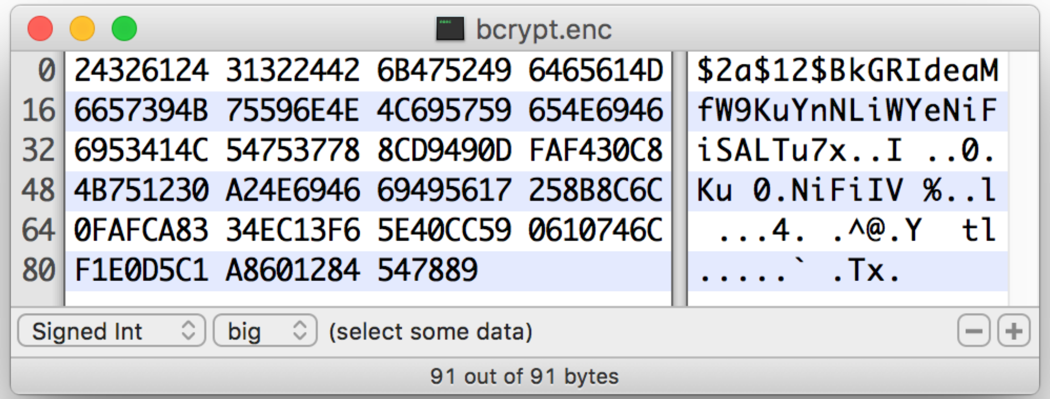
OpenSSL allows for salted or unsalted key derivation. *Unsalted key derivation is a security risk and is not recommended.* If a salt is present, the first 8 bytes of the input are the ASCII string "Salted__" (0x53 61 6C 74 65 64 5F 5F) and the next 8 bytes are the ASCII-encoded salt. On decryption, the salt is read in and combined with the password to derive the encryption key and IV. If there is no salt header, the entire input is considered to be the cipher text.



```
openssl_evp.enc
 0 53616C74 65645F5F AABBCCDD EEFF0011  Salted__......
16 47972607 296F80B9 9816D205 9BF899F4  G.& )o... . ....
32 8E8EB8DF F37DAC28 370DA3A6 05865894  .....}.(7 .. .X.
48
Signed Int   big   (select some data)
16 out of 48 bytes
```

For new KDFs, each of which allow for non-deterministic IVs, the IV must be stored alongside the cipher text. This is not a vulnerability, as the IV is not required to be secret, but simply to be unique for messages encrypted using the same key to reduce the success of cryptographic attacks. For these KDFs, the output consists of the salt, followed by the salt delimiter, UTF-8 string "NiFiSALT" (0x4E 69 46 69 53 41 4C 54) and then the IV, followed by the IV delimiter, UTF-8 string "NiFiIV" (0x4E 69 46 69 49 56), followed by the cipher text.

Bcrypt, Scrypt, PBKDF2

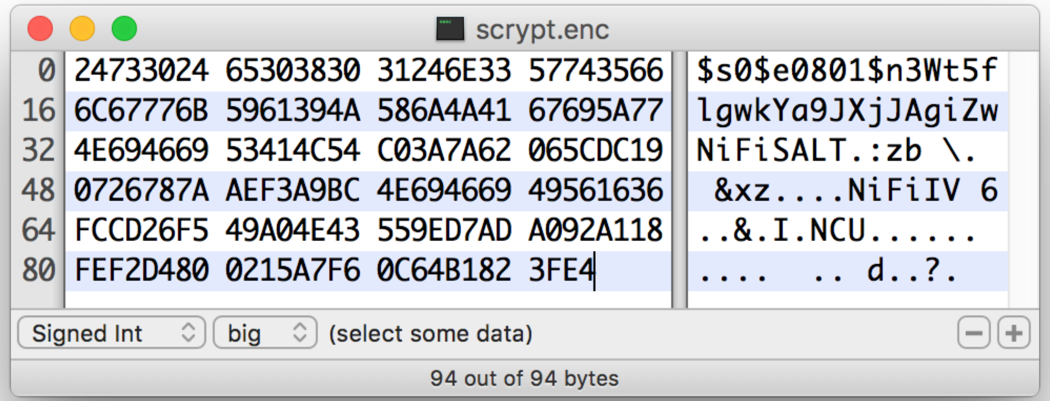


bcrypt.enc

```
 0 24326124 31322442 6B475249 6465614D  $2a$12$BkGRIdeaM
16 6657394B 75596E4E 4C695759 654E6946  fW9KuYnNLiWYeNiF
32 6953414C 54753778 8CD9490D FAF430C8  iSALTu7x..I ..0.
48 4B751230 A24E6946 69495617 258B8C6C  Ku 0.NiFiIV %..l
64 0FAFCA83 34EC13F6 5E40CC59 0610746C   ...4. .^@.Y  tl
80 F1E0D5C1 A8601284 547889             .....` .Tx.
```

Signed Int | big | (select some data)

91 out of 91 bytes

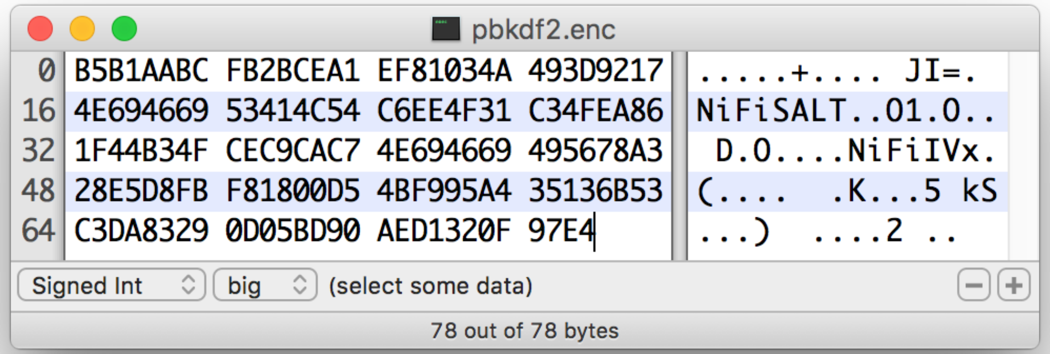


scrypt.enc

```
 0 24733024 65303830 31246E33 57743566  $s0$e0801$n3Wt5f
16 6C67776B 5961394A 586A4A41 67695A77  lgwkYa9JXjJAgiZw
32 4E694669 53414C54 C03A7A62 065CDC19  NiFiSALT.:zb \.
48 0726787A AEF3A9BC 4E694669 49561636   &xz....NiFiIV 6
64 FCCD26F5 49A04E43 559ED7AD A092A118  ..&.I.NCU......
80 FEF2D480 0215A7F6 0C64B182 3FE4      ....   .. d..?.
```

Signed Int | big | (select some data)

94 out of 94 bytes



```
 0  B5B1AABC FB2BCEA1 EF81034A 493D9217  .....+.... JI=.
16  4E694669 53414C54 C6EE4F31 C34FEA86  NiFiSALT..O1.O..
32  1F44B34F CEC9CAC7 4E694669 495678A3   D.O....NiFiIVx.
48  28E5D8FB F81800D5 4BF995A4 35136B53  (....  .K...5 kS
64  C3DA8329 0D05BD90 AED1320F 97E4      ...)  ....2 ..
```

Java Cryptography Extension (JCE) Limited Strength Jurisdiction Policies

Because of US export regulations, default JVMs have http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html#importlimits available to them. For example, AES operations are limited to 128 bit keys by default. While AES-128 is cryptographically safe, this can have unintended consequences, specifically on Password-based Encryption (PBE).

PBE is the process of deriving a cryptographic key for encryption or decryption from user-provided secret material, usually a password. Rather than a human remembering a (random-appearing) 32 or 64 character hexadecimal string, a password or passphrase is used.

A number of PBE algorithms provided by NiFi impose strict limits on the length of the password due to the underlying key length checks. Below is a table listing the maximum password length on a JVM with limited cryptographic strength.

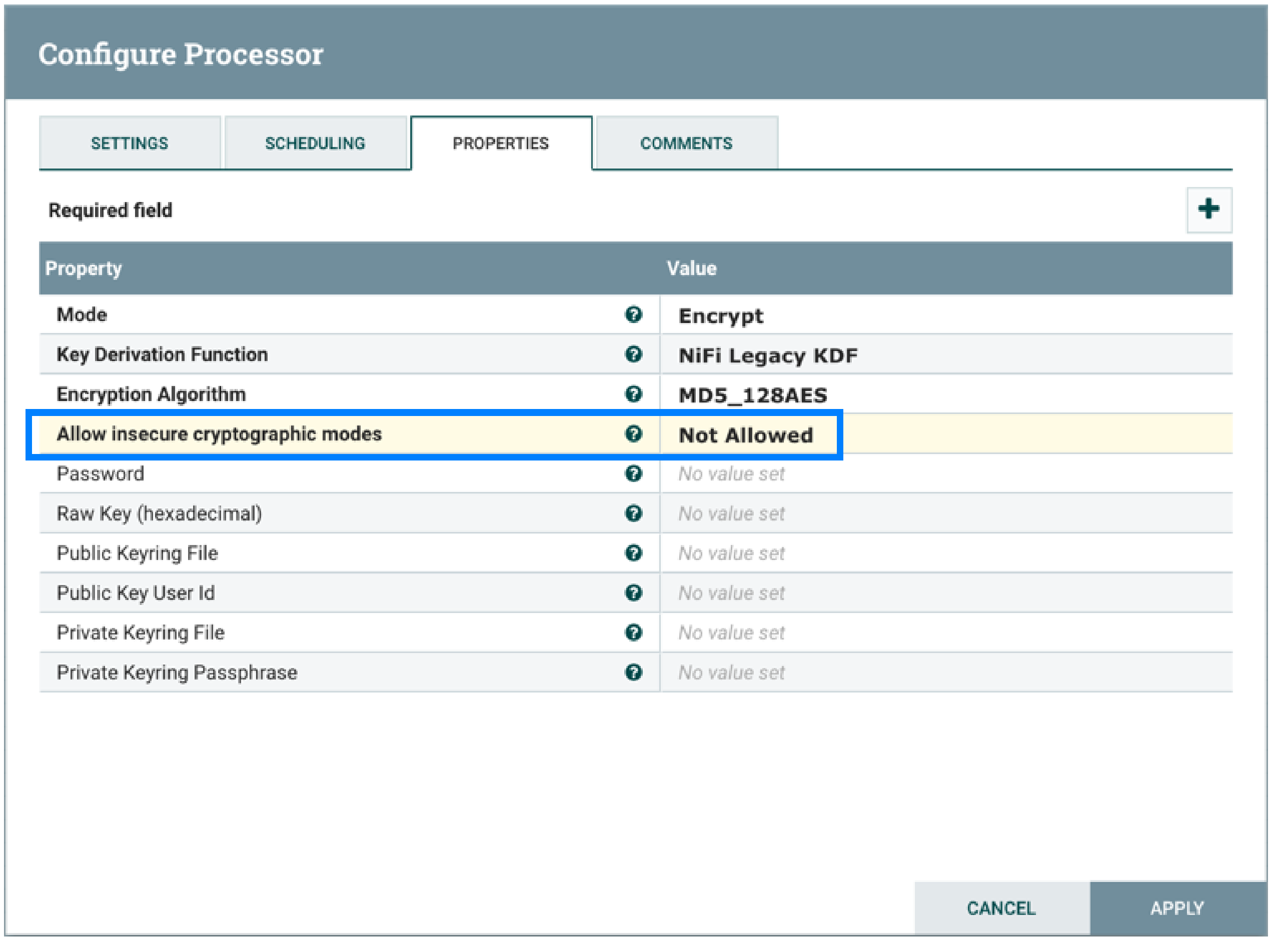
Table 1. Maximum Password Length on Limited Cryptographic Strength JVM

Algorithm	**Max Password Length**
PBEWITHMD5AND128BITAES-CBC-OPENSSL	16
PBEWITHMD5AND192BITAES-CBC-OPENSSL	16
PBEWITHMD5AND256BITAES-CBC-OPENSSL	16
PBEWITHMD5ANDDES	16
PBEWITHMD5ANDRC2	16
PBEWITHSHA1ANDRC2	16
PBEWITHSHA1ANDDES	16
PBEWITHSHAAND128BITAES-CBC-BC	7
PBEWITHSHAAND192BITAES-CBC-BC	7
PBEWITHSHAAND256BITAES-CBC-BC	7
PBEWITHSHAAND40BITRC2-CBC	7

Algorithm	Max Password Length
PBEWITHSHAAND128BITRC2-CBC	7
PBEWITHSHAAND40BITRC4	7
PBEWITHSHAAND128BITRC4	7
PBEWITHSHA256AND128BITAES-CBC-BC	7
PBEWITHSHA256AND192BITAES-CBC-BC	7
PBEWITHSHA256AND256BITAES-CBC-BC	7
PBEWITHSHAAND2-KEYTRIPLEDES-CBC	7
PBEWITHSHAAND3-KEYTRIPLEDES-CBC	7
PBEWITHSHAANDTWOFISH-CBC	7

Allow Insecure Cryptographic Modes

By default, the Allow Insecure Cryptographic Modes property in EncryptContent processor settings is set to not-allowed. This means that if a password of fewer than 10 characters is provided, a validation error will occur. 10 characters is a conservative estimate and does not take into consideration full entropy calculations, patterns, etc.



Property	Value
Mode	Encrypt
Key Derivation Function	NiFi Legacy KDF
Encryption Algorithm	MD5_128AES
Allow insecure cryptographic modes	Not Allowed
Password	No value set
Raw Key (hexadecimal)	No value set
Public Keyring File	No value set
Public Key User Id	No value set
Private Keyring File	No value set
Private Keyring Passphrase	No value set

On a JVM with limited strength cryptography, some PBE algorithms limit the maximum password length to 7, and in this case it will not be possible to provide a "safe" password. It is recommended to install the JCE Unlimited Strength Jurisdiction Policy files for the JVM to mitigate this issue.

- http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

If on a system where the unlimited strength policies cannot be installed, it is recommended to switch to an algorithm that supports longer passwords (see table above).

Allowing Weak Crypto

If it is not possible to install the unlimited strength jurisdiction policies, the Allow Weak Crypto setting can be changed to allowed, but this is not recommended. Changing this setting explicitly acknowledges the inherent risk in using weak cryptographic configurations.

It is preferable to request upstream/downstream systems to switch to https://cwiki.apache.org/confluence/display/NIFI/Encryption+Information or use a "strong" https://cwiki.apache.org/confluence/display/NIFI/Key+Derivation+Function+Explanations.

Encrypted Passwords in Configuration Files

In order to facilitate the secure setup of NiFi, you can use the encrypt-config command line utility to encrypt raw configuration values that NiFi decrypts in memory on startup. This extensible protection scheme transparently allows NiFi to use raw values in operation, while protecting them at rest. In the future, hardware security modules (HSM) and external secure storage mechanisms will be integrated, but for now, an AES encryption provider is the default implementation.

This is a change in behavior; prior to 1.0, all configuration values were stored in plaintext on the file system. POSIX file permissions were recommended to limit unauthorized access to these files

If no administrator action is taken, the configuration values remain unencrypted.

Encrypt-Config Tool

The encrypt-config command line tool (invoked as ./bin/encrypt-config.sh or bin\encrypt-config.bat) reads from a 'nifi.properties' file with plaintext sensitive configuration values, prompts for a master password or raw hexadecimal key, and encrypts each value. It replaces the plain values with the protected value in the same file, or writes to a new 'nifi.properties' file if specified.

The default encryption algorithm utilized is AES/GCM 128/256-bit. 128-bit is used if the JCE Unlimited Strength Cryptographic Jurisdiction Policy files are not installed, and 256-bit is used if they are installed.

You can use the following command line options with the encrypt-config tool:

- -h,--help Prints this usage message
- -v,--verbose Sets verbose mode (default false)
- -n,--niFiProperties <arg> The nifi.properties file containing unprotected config values (will be overwritten)
- -l,--loginIdentityProviders <arg> The login-identity-providers.xml file containing unprotected config values (will be overwritten)
- -a,--authorizers <arg> The authorizers.xml file containing unprotected config values (will be overwritten)
- -f,--flowXml <arg> The flow.xml.gz file currently protected with old password (will be overwritten)
- -b,--bootstrapConf <arg> The bootstrap.conf file to persist master key
- -o,--outputNiFiProperties <arg> The destination nifi.properties file containing protected config values (will not modify input nifi.properties)
- -i,--outputLoginIdentityProviders <arg> The destination login-identity-providers.xml file containing protected config values (will not modify input login-identity-providers.xml)

- -u,--outputAuthorizers <arg> The destination authorizers.xml file containing protected config values (will not modify input authorizers.xml)
- -g,--outputFlowXml <arg> The destination flow.xml.gz file containing protected config values (will not modify input flow.xml.gz)
- -k,--key <arg> The raw hexadecimal key to use to encrypt the sensitive properties
- -e,--oldKey <arg> The old raw hexadecimal key to use during key migration
- -p,--password <arg> The password from which to derive the key to use to encrypt the sensitive properties
- -w,--oldPassword <arg> The old password from which to derive the key during migration
- -r,--useRawKey If provided, the secure console will prompt for the raw key value in hexadecimal form
- -m,--migrate If provided, the nifi.properties and/or login-identity-providers.xml sensitive properties will be re-encrypted with a new key
- -x,--encryptFlowXmlOnly If provided, the properties in flow.xml.gz will be re-encrypted with a new key but the nifi.properties and/or login-identity-providers.xml files will not be modified
- -s,--propsKey <arg> The password or key to use to encrypt the sensitive processor properties in flow.xml.gz
- -A,--newFlowAlgorithm <arg> The algorithm to use to encrypt the sensitive processor properties in flow.xml.gz
- -P,--newFlowProvider <arg> The security provider to use to encrypt the sensitive processor properties in flow.xml.gz

As an example of how the tool works, assume that you have installed the tool on a machine supporting 256-bit encryption and with the following existing values in the 'nifi.properties' file:

```
# security properties #
nifi.sensitive.props.key=thisIsABadSensitiveKeyPassword
nifi.sensitive.props.algorithm=PBEWITHMD5AND256BITAES-CBC-OPENSSL
nifi.sensitive.props.provider=BC
nifi.sensitive.props.additional.keys=

nifi.security.keystore=/path/to/keystore.jks
nifi.security.keystoreType=JKS
nifi.security.keystorePasswd=thisIsABadKeystorePassword
nifi.security.keyPasswd=thisIsABadKeyPassword
nifi.security.truststore=
nifi.security.truststoreType=
nifi.security.truststorePasswd=
```

Enter the following arguments when using the tool:

```
encrypt-config.sh
-b bootstrap.conf
-k 0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA9876543210
-n nifi.properties
```

As a result, the 'nifi.properties' file is overwritten with protected properties and sibling encryption identifiers (aes/gcm/256, the currently supported algorithm):

```
# security properties #
nifi.sensitive.props.key=n2z+tTTbHuZ4V4V2||uWhdasyDXD4ZG2lMAes/
vqh6u4vaz4xgL4aEbF4Y/dXevqk3ulRcOwf1vc4RDQ==
nifi.sensitive.props.key.protected=aes/gcm/256
nifi.sensitive.props.algorithm=PBEWITHMD5AND256BITAES-CBC-OPENSSL
nifi.sensitive.props.provider=BC
nifi.sensitive.props.additional.keys=

nifi.security.keystore=/path/to/keystore.jks
nifi.security.keystoreType=JKS
nifi.security.keystorePasswd=oBjT92hIGRElIGOh||MZ6uYuWNBrOA6usq/
Jt3DaD2e4otNirZDytac/w/KFe0HOkrJR03vcbo
nifi.security.keystorePasswd.protected=aes/gcm/256
nifi.security.keyPasswd=ac/BaE35SL/esLiJ||
+ULRvRLYdIDA2VqpE0eQXDEMjaLBMG2kbKOdOwBk/hGebDKlVg==
nifi.security.keyPasswd.protected=aes/gcm/256
nifi.security.truststore=
nifi.security.truststoreType=
nifi.security.truststorePasswd=
```

Additionally, the 'bootstrap.conf' file is updated with the encryption key as follows:

```
# Master key in hexadecimal format for encrypted sensitive configuration
 values
nifi.bootstrap.sensitive.key=0123456789ABCDEFFEDCBA98765432100123456789ABCDEFFEDCBA98765
```

Sensitive configuration values are encrypted by the tool by default, however you can encrypt any additional properties, if desired. To encrypt additional properties, specify them as comma-separated values in the nifi.sensitive.props.additional.keys property.

If the 'nifi.properties' file already has valid protected values, those property values are not modified by the tool.

When applied to 'login-identity-providers.xml' and 'authorizers.xml', the property elements are updated with an encryption attribute:

Example of protected login-identity-providers.xml:

```
   <!-- LDAP Provider -->
   <provider>
       <identifier>ldap-provider</identifier>
       <class>org.apache.nifi.ldap.LdapProvider</class>
       <property name="Authentication Strategy">START_TLS</property>
       <property name="Manager DN">someuser</property>
       <property name="Manager Password" encryption="aes/
gcm/128">q4r7WIgN0MaxdAKM||SGgdCTPGSFEcuH4RraMYEdeyVbOx93abdWTVSWvh1w+klA</
property>
       <property name="TLS - Keystore"></property>
       <property name="TLS - Keystore Password" encryption="aes/
gcm/128">Uah59TWX+Ru5GY5p||B44RT/LJtC08QWA5ehQf01JxIpf0qSJUzug25UwkF5a50g</
property>
       <property name="TLS - Keystore Type"></property>
       ...
   </provider>
```

Example of protected authorizers.xml:

```
   <!-- LDAP User Group Provider -->
   <userGroupProvider>
       <identifier>ldap-user-group-provider</identifier>
       <class>org.apache.nifi.ldap.tenants.LdapUserGroupProvider</class>
       <property name="Authentication Strategy">START_TLS</property>
       <property name="Manager DN">someuser</property>
       <property name="Manager Password" encryption="aes/
gcm/128">q4r7WIgN0MaxdAKM||SGgdCTPGSFEcuH4RraMYEdeyVbOx93abdWTVSWvh1w+klA</
property>
       <property name="TLS - Keystore"></property>
       <property name="TLS - Keystore Password" encryption="aes/
gcm/128">Uah59TWX+Ru5GY5p||B44RT/LJtC08QWA5ehQf01JxIpf0qSJUzug25UwkF5a50g</
property>
       <property name="TLS - Keystore Type"></property>
       ...
   </userGroupProvider>
---
```

Sensitive Property Key Migration

In order to change the key used to encrypt the sensitive values, indicate migration mode using the -m or --migrate flag, provide the new key or password using the -k or -p flags as usual, and provide the existing key or password using -e or -w respectively. This will allow the toolkit to decrypt the existing values and re-encrypt them, and update bootstrap.conf with the new key. Only one of the key or password needs to be specified for each phase (old vs. new), and any combination is sufficient:

- old key # new key
- old key # new password
- old password # new key
- old password # new password

Existing Flow Migration

This tool can also be used to change the value of nifi.sensitive.props.key for an existing flow. The tool will read the existing flow.xml.gz and decrypt any sensitive component properties using the original key, then re-encrypt the sensitive properties with the new key, and write out a new version of the flow.xml.gz, or overwrite the existing one.

The current sensitive properties key is not provided as a command-line argument, as it is read directly from nifi.properties. As this file is a required parameter, the -x/--encryptFlowXmlOnly flags tell the tool not to attempt to encrypt the properties in nifi.properties, but rather to only update the nifi.sensitive.props.key value with the new key. The exception to this is if the nifi.properties is already encrypted, the new sensitive property key will also be encrypted before being written to nifi.properties.

The following command would migrate the sensitive properties key in place, meaning it would overwrite the existing flow.xml.gz and nifi.properties:

```
./encrypt-config.sh -f /path/to/flow.xml.gz -n ./path/to/nifi.properties -s
 newpassword -x
```

The following command would migrate the sensitive properties key and write out a separate flow.xml.gz and nifi.properties:

```
./encrypt-config.sh -f ./path/to/src/flow.xml.gz -g /path/to/dest/
flow.xml.gz -n /path/to/src/nifi.properties -o /path/to/dest/nifi.properties
 -s newpassword -x
```

Password Key Derivation

Instead of providing a 32 or 64 character raw hexadecimal key, you can provide a password from which the key will be derived. As of 1.0.0, the password must be at least 12 characters, and the key will be derived using SCrypt with the parameters:

- pw - the password bytes in UTF-8
- salt - the fixed salt value (NIFI_SCRYPT_SALT) bytes in UTF-8
- N - 216
- r - 8
- p - 1
- dkLen - determined by the JCE policies available

As of August 2016, these values are determined to be strong for this threat model but may change in future versions.

Note: While fixed salts are counter to best practices, a static salt is necessary for deterministic key derivation without additional storage of the salt value.

Secure Prompt

If you prefer not to provide the password or raw key in the command-line invocation of the tool, leaving these arguments absent will prompt a secure console read of the password (by default) or raw key (if the -r flag is provided at invocation).

Kerberos Service

NiFi can be configured to use Kerberos SPNEGO (or "Kerberos Service") for authentication. In this scenario, users will hit the REST endpoint /access/kerberos and the server will respond with a 401 status code and the challenge response header WWW-Authenticate: Negotiate. This communicates to the browser to use the GSS-API and load the user's Kerberos ticket and provide it as a Base64-encoded header value in the subsequent request. It will be of the form Authorization: Negotiate YII…. NiFi will attempt to validate this ticket with the KDC. If it is successful, the user's principal will be returned as the identity, and the flow will follow login/credential authentication, in that a JWT will be issued in the response to prevent the unnecessary overhead of Kerberos authentication on every subsequent request. If the ticket cannot be validated, it will return with the appropriate error response code. The user will then be able to provide their Kerberos credentials to the login form if the KerberosLoginIdentityProvider has been configured.

NiFi will only respond to Kerberos SPNEGO negotiation over an HTTPS connection, as unsecured requests are never authenticated.

The following properties must be set in nifi.properties to enable Kerberos service authentication.

Property	Required	Description
Service Principal	true	The service principal used by NiFi to communicate with the KDC
Keytab Location	true	The file path to the keytab containing the service principal

Notes

- Kerberos is case-sensitive in many places and the error messages (or lack thereof) may not be sufficiently explanatory. Check the case sensitivity of the service principal in your configuration files. Convention is HTTP/fully.qualified.domain@REALM.
- Browsers have varying levels of restriction when dealing with SPNEGO negotiations. Some will provide the local Kerberos ticket to any domain that requests it, while others whitelist the trusted domains.
- Some browsers (legacy IE) do not support recent encryption algorithms such as AES, and are restricted to legacy algorithms (DES). This should be noted when generating keytabs.
- The KDC must be configured and a service principal defined for NiFi and a keytab exported. Comprehensive instructions for Kerberos server configuration and administration are beyond the scope of this document, but an example is below:

Adding a service principal for a server at nifi.nifi.apache.org and exporting the keytab from the KDC:

```
root@kdc:/etc/krb5kdc# kadmin.local
Authenticating as principal admin/admin@NIFI.APACHE.ORG with password.
kadmin.local:  listprincs
K/M@NIFI.APACHE.ORG
admin/admin@NIFI.APACHE.ORG
```

```
...
kadmin.local:  addprinc -randkey HTTP/nifi.nifi.apache.org
WARNING: no policy specified for HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG;
 defaulting to no policy
Principal "HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG" created.
kadmin.local:  ktadd -k /http-nifi.keytab HTTP/nifi.nifi.apache.org
Entry for principal HTTP/nifi.nifi.apache.org with kvno 2, encryption type
 des3-cbc-sha1 added to keytab WRFILE:/http-nifi.keytab.
Entry for principal HTTP/nifi.nifi.apache.org with kvno 2, encryption type
 des-cbc-crc added to keytab WRFILE:/http-nifi.keytab.
kadmin.local:  listprincs
HTTP/nifi.nifi.apache.org@NIFI.APACHE.ORG
K/M@NIFI.APACHE.ORG
admin/admin@NIFI.APACHE.ORG
...
kadmin.local: q
root@kdc:~# ll /http*
-rw------- 1 root root 162 Mar 14 21:43 /http-nifi.keytab
root@kdc:~#
```