Adding a new EAP method to the UI

By, Chris Hessing (chris@open1x.org)

The purpose of this document is to provide developers with a road map on how to add new EAP methods to the XSupplicant UI. This document follows my implementation of EAP-FAST in the UI.

Implementing the Wizard

The first thing you need to do is look at the configuration options that are used in your EAP method to determine if you will need any additional wizard widgets to fully configure your EAP method. EAP-FAST is similar in a lot of ways to EAP-PEAP, or EAP-TTLS. So looking at the widgets used for those methods is a good way to start to line up the configuration options you might need.

It is not uncommon for EAP method implementations to have configuration options that a normal user wouldn't use. Identify those options, and ignore them for now. It is also possible that even though some options are yes/no values and can be mixed, a normal user wouldn't. EAP-FAST is a perfect example of both of these situations. A normal user would never disable PAC provisioning, so there is no need to expose a configuration option for that in the wizard. (Though, it may make sense to expose it in the advanced configuration window.) Also, even though it is permissible to configure EAP-FAST to allow both authenticated provisioning, and anonymous provisioning, in practice this would be an unusual configuration, so in the wizard, we will want to make them mutually exclusive with the use of radio buttons.

EAP-FAST's configuration is very similar to the configuration of EAP-PEAP, so looking at how EAP-PEAP is implemented in the wizard is a good place to start. The PEAP wizard has all of the pages that we will need to complete a configuration of EAP-FAST, and they are already in a logical order. However, EAP-FAST has the following extra items that we will need to configure :

<Allow_Anonymous_Provision> and <Allow_Authenticated_Provision>

As I mentioned earlier, we want to make these options radio buttons, so we are going to need to make a custom widget for EAP-FAST in order to add these options. The easiest way to do that is to make a copy of the existing widget used for EAP-PEAP configuration, and modify it. So, we make a copy of wizardPageDot1XInnerProtocol.ui, and call it wizardPageFASTInnerProtocol.ui. Once we have done this, we can open wizardPageFASTInnerProtocol.ui and make our changes. Specifically, we want to put radio buttons around the "Validate server certificate" checkbox. Above the checkbox, we want to add "Use authenticated provisioning", and below it we want to add "Use anonymous provisioning". Also, the

“Validate server certificate” checkbox now only has meaning if “Use authenticated provisioning” is selected, so we want to indent that a little to show this.

NOTE *: Do not change the size of the form. This form will be used in a widget stack, and changing its size will have unexpected results!*

Now that we have finished our UI widget, we can save it, and move on to writing the code to support it.

In the supplicant’s configuration file (and in the engine) the configuration data is stored in a hierarchical fashion. In the UI, it is stored in a flat fashion, and converted when it is saved or read. Since we are adding configuration options for the two new settings, we need to add variables to store that information in. In ConnectionWizardData.h, we want to add two new private variables to the class. So, we want to change this :

```
QString m_SCreader;
bool m_autoRealm;
bool m_validateCert;
```

to this :

```
QString m_SCreader;
bool m_autoRealm;
bool m_anonymousProvisioning;
bool m_authenticatedProvisioning;
bool m_validateCert;
```

Next, we need to let the class know that there is a new EAP type that can be configured. Since this is a phase 1 EAP method, we need to add it to the phase 1 EAP method enumerations. Find this block above the variables you just created :

```
// 802.1X settings
typedef enum {
      eap_peap,
      eap_ttls,
      eap_aka,
      eap_sim,
      eap_md5
} Dot1XProtocol;
```

And change it to :

```
// 802.1X settings
typedef enum {
      eap_peap,
      eap_ttls,
      eap_aka,
      eap_sim,
      eap_fast,
```

```
        eap_md5
    } Dot1XProtocol;
```

Notice that we now have an eap_fast option in the enumeration. Where you add new values to the list is irrelevant, as long as you refer to members of the enumeration by name, and not number.

Now, we need to edit the code that reads the configuration from the supplicant engine, and writes the information back. The easiest way to do this is to use a prototype that is similar to what we are adding. Since we have been using PEAP as our template so far, we will continue to use it by copying this prototype :

```
bool toProfileEAP_PEAPProtocol(config_profiles * const, config_trusted_server
const * const);
```

and creating this prototype :

```
bool toProfileEAP_FASTProtocol(config_profiles * const, config_trusted_server
const * const);
```

This function will be used to save the configuration information back to the supplicant engine when the user instructs the UI to do so. We don't need to create a function for reading the data, since it is all handled in the initFromSupplicantProfiles() function.

At this point, we should be done editing the ConnectionWizardData.h file. Save it, and open up the ConnectionWizardData.cpp file to implement the toProfileEAP_FASTProtocol() function.

There are two ways to implement this function. One is to write it completely from scratch. The other is to copy an existing function, and modify it fit the needs of our new EAP type. Since we have been using PEAP as a template, we will continue to do that. Make a copy of toProfileEAP_PEAPProtocol(), rename it to toProfileEAP_FASTProtocol(), and go through it changing things from referencing PEAP to referencing FAST.

Changing the function involves changing references to EAP_TYPE_PEAP, and config_eap_peap to EAP_TYPE_FAST, and config_eap_fast. (It also makes sense to change variables such as "mypeap" to something that will make sense like "myfast".)

Once you have done that, you need to find any variables that existing in the PEAP implementation that are not in FAST, and remove them. In the places that you remove those variables, you will want to replace them with configuration variables for your EAP method. For our EAP-FAST implementation we want to remove "mypeap->force_peap_version" and replace it with our new configuration options for FAST.

Rather than copy all of the code changes here, I would suggest looking at the implementations for toProfileEAP_PEAPProtocol() and toProfileEAP_FASTProtocol() to see what I changed. Following this step, you need to go in to the toProfileData() function, and add your EAP method to the case statement there so that the UI knows the proper way of saving the configuration.

Once you have completed your changes on toProfileEAP_FASTProtocol(), it is time to put the code in place to parse the configuration data and store it in the right variables. To do that, go in to the initFromSupplicantProfiles() function, find the section for PEAP, and copy it. Then, go through and make your changes to fill in the needed variables based on the information in your configuration structure. For an example, look at the EAP-FAST section in initFromSupplicantProfiles() and compare it to the EAP-PEAP section.

You should now be finished editing the ConnectionWizardData.cpp file. Now, we need to actually use the configuration widget we used, and configure the code to know how to go forward and backward through the wizard. To do that, open WizardPages.cpp, and WizardPages.h.

The WizardPage base class has all of the members that you should override to implement a new page. Because the page I am implementing it very similar to the WizardPageDot1XInnerProtocol class, I am using a copy of that class to start with. To that class, I need to add the radio button objects for the radio buttons that I added to the form at the beginning of this document. Once I have done that, I am ready to go to the WizardPages.cpp file, and create my implementation.

Start by going to WizardPageDot1XProtocol class and add your EAP method name to the create() function. After that, you need to go to the init() function, and add your EAP method name and add a line to the switch statement that will map your EAP method name to its index in the combo box. (Failing to do this will result in the wrong EAP method being selected in the combo box when using the wizard to edit a configuration.) Finally, go to the wizardData() function, and add code to the switch statement to map the combo boxes index back to the appropriate EAP method.

In the create() member of your class, you need to load the widget from the file on disk. I suggest looking at other create() members to determine how that is done. Once you have loaded the form, you need to map the objects on that form to objects that you can work with. This mapping is done using the names that were given to the objects on the form when you created it. It is often easiest to have the form open while you work through this part.

Once your create() member is complete, you need to set up your init() member. The init() member should set any objects on the form to show the data that was read from the code you added to ConnectionWizardData class. It is often a good idea to check and make sure the object you are setting is valid before you set data to it. One of the goals of allowing the forms to be loaded from disk is people can modify them to fit their own needs. As a result, the form may not have all of the objects on it that you originally put there.

Now, you need to implement your wizardData() member. This member should take the data that is in the forms, and store it to the ConnectionWizardData class so that it can later be sent to the supplicant engine, and saved. Take special care in this function not to miss any important variables, or you will end up wondering why some of your configuration settings don’t get saved.

Many wizard page widgets would be complete at this point. However, our form has a radio button that when it is in one state we want to disable the “Verify certificate” check box. So, we need to implement a set of slots/signals to make sure that we change the state of those widgets.

At this point, we have almost everything we need in order to configure out new EAP method. The last thing we need to do is create the path that the wizard will follow. To do this, you want to hop over to ConnectionWizard.h. Locate the wizardPages enumeration, and add a name for your new wizard page. I called mine “pageFastInnerProtocol” to make it obvious what it is supposed to do. Then, open up ConnectionWizard.cpp where we will add the final pieces to make the wizard work correctly.

Our first stop should be the loadPages() function. This function is where you instantiate the page which will be inserted in the wizard’s widget stack. In the case statement, add the enumeration that you put in ConnectionWizard.h, and have it instantiate the class that you created in WizardPages.cpp.

Next, scroll down to getNextPage() and add your enumeration value to the case statement in this function. The case statement that you create will define the page that the user will be displayed when they click the “Next” button. So if “Next” takes you to a page you didn’t expect, this is where to look. In addition to creating your case statement for your new page, you will also need to edit the case statement for the page that comes before it so that you get directed to the proper page.

Assuming you did everything correctly, you should now be able to build the UI, and configure your EAP method through the wizard. Next, we will go through how to add it to the advanced configuration window.