Antoine Bellanger

Disclaimer: I advise you to carefully read all the blog post. I give a lot of information and copying only the code will result in a failure or even worse as we are dealing with very fragile components. I personally had to delete everything I had on my Mac more than a few times because of weird bugs. You should also be comfortable with SSH, Shell scripts and command line.

Disclaimer #2: I made every attempt to ensure the accuracy and reliability of this blog post but code evolves so the information is provided "as is". I won't accept any responsibility or liability in case of problems with the code.

What is this "Plugin View"?

The official definition of an *SFAuthorizationPluginView* says it:

"Allows authorization plug-in developers to create a custom view their plug-in can display." - Apple Developer

The definition is quite exhaustive so let's be a little bit more detailed and clear.

An *SFAuthorizationPluginView* allows you to create a custom UI component that will be displayed on the login window. It is useful if you want to let the user login with a system that you engineered.

For example, you could store the user's password inside the code (this is not safe - I don't recommend doing this) and instead show a plugin asking the user to pick a color. If the color is right, log the user in.

Let's start

Apple used to have a demo project on their website but it seems they have

removed it. It's not too bad as it was not working anymore. I used a copy of this project which was being maintained but was abandoned since five years so I updated it myself.

You can clone it directly from my Github.

The Code

I won't dive deep into the code but just a word on the main files.

AuthPlugin.h / AuthPlugin.m

This part of the code is responsible for the *invoke* mechanism that will be called from the *loginwindow*. It initializes the plugin view that is coded in the next part. There's not much to change and play with here for the demo.

If you want to have different mechanisms identifiers initializing different plugins or methods, you can do that from here.

NameAndPasswordPlugin.h / NameAndPasswordPlugin.m

Here you can create the links with your UI and handle all the authorization mechanisms. This is where you are going to be able to code whatever the login process you want to appear.

SSH Connection

I will not introduce in detail what is SSH here but it is simply a protocol which allows you to communicate between your computers.

SSH connection is useful when you test your *SFAuthorizationPluginView* because you can access the Terminal of the Mac which will be showing the plugin from another Mac. Therefore, you can try saving it if something is broken with your plugin before being forced to reboot it.

On the Mac which will havethe plugin installed, go to *Sharing* in the *System*

Preferences and there enable *Remote Login*. You should have a line saying "To log in to this computer remotely, type *"ssh 'session'@'mac[or]ip'"*.

Just type in this command into the Terminal of your secondary Mac and follow the instructions. You should be asked about the authenticity of the fingerprint (say yes) and to type in the password to log into your Mac.

There, on the Desktop (or where you want), create a folder like "SFTesting". We will use it after to store the backup and the updated necessary files.

Installation & Testing

Plugin Installation

Obviously, you will need to build the project to start.

Then, there are a few things to do to install the plugin.

1. Copy the *bundle* in the */SecurityAgentPlugins* folder.
2. Change the ownership to root.
3. Set permissions to *rwxr-xr-x*.

1. Copy the *bundle* in the SecurityAgentPlugins folder.

Where can you find the *bundle*?

Xcode stores all the builds in a folder named DerivedData. You can access it by typing into the Terminal:

cd ~/Library/Developer/Xcode/DerivedData/NameAndPassword-XXX/Build/Products/Debug

where *XXX* is a random sequence created by Xcode.

In this folder, you should have the latest build of your *NameAndPassword.bundle*. You will need to copy it to

/Library/Security/SecurityAgentPlugins. You can do that by using the *cp* command:

sudo cp -R NameAndPassword.bundle /Library/Security/SecurityAgentPlugins/

Note: Sometimes, for a weird reason I haven't find yet, this command wouldn't copy correctly the latest *NameAndPassword.bundle* to the folder. What I did as a workaround was to change the version number with something like 1.0.XXX every time I created a new build so I could check if the latest version was in the */SecurityAgentPlugins* folder (simply using the details showed in the Finder).

2. Change the ownership to root.

Simply type in this command:

sudo chown -R root:wheel /Library/Security/SecurityAgentPlugins/NameAndPassword.bundle

3. Set permissions to rwxr-xr-x (755).

Same, type in this command in your Terminal:

sudo chmod -R 755 /Library/Security/SecurityAgentPlugins/NameAndPassword.bundle

Now the plugin is installed where it should be, has the right permissions and ownership.

Loading the plugin

We now need to update the *security.login.console* so that it loads our custom plugin.

Here we need to be very careful about the modifications we are changing a

core part of the login system. My advice would be to double check every time all the parameters if you want to avoid having to erase your hard drive and install a clean copy of macOS.

The *security.login.console* is a plist file that contains the mechanisms that are called when the *loginwindow* is showed.

First, let's save the current procedure, for this, you need to run:

cd ~/Desktop/SFTesting (or where you stored your folder earlier)

security authorizationdb read system.login.console > system.login.console.original.txt

You will get a *system.login.console.original.txt* containing the current mechanisms. We are going to make a copy of this file so we keep the original as a backup in case we need to restore it. It is really important to keep this file as is. It will be necessary if the login system fails to load the plugin system.

cp system.login.console.original.txt system.login.console.custom.txt

Let's edit the custom file so that during login the system invokes the NameAndPassword plugin. Using your favorite text editor (*vi*, TextEdit...), you need to change the line *loginwindow:login* with *NameAndPassword:invoke*. (Please make sure that you copied this exactly. Be careful also about the ":".)

This means that instead of showing the classical *loginwindow*, it will instead load our custom plugin view.

Then, let's update the *authorizationdb* with our updated mechanisms :

sudo security authorizationdb write system.login.console < system.login.console.custom.txt

That's it, the plugin is ready to use. You should now establish an

SSH connection between your two Mac. You will be able to try and restore the login system from the other Mac in case the plugin is broken and crashes the login system.

And you're all set! Now if you log out (> Log Out) or switch users (see the *suspend.sh* script at the end of this post), you should see the NameAndPassword custom UI! Beware, "Sleep" won't be working as it is using another mechanisms file which is *security.login.screensaver*.

Handling Bugs & Crashes

Here we go, you made a modification to your code, the *loginwindow* doesn't reload and you're stuck on a black screen, you have an infinite loading spinner or a crashing loading spinner. Nothing really funny.

What can you do? Mainly try to restore the original login system using the SSH connection.

If it is not done already, on your Mac #2, *cd* to the folder where are your *security.login.console.txt* files you saved earlier. And here type :

security authorizationdb write system.login.console < system.login.console.original.txt

Add directly after:

sudo killall SecurityAgent

sudo killall securityd

This will put instead of your custom mechanisms file the original one and will help the login system to reload. If nothing happens but you have access to the faulty custom login, make it crash and with those commands, it will reboot in its original version.

If it doesn't work? Turn off the Mac and reboot it normally. It will load a

normal *system.login.console*.

Still doesn't work ? Then it doesn't look good. Try rebooting in Safe Mode and hope it will work. Otherwise you will need to install a clean version of macOS.

A few remarks

Here are a few advice and things I noticed that during all my experimentations.

- When the plugin loads, it will not be able to update UI on a regular (or irregular basis). That means if you need to update the UI and want to see the change, you need to go back to the view showing all the users and click again on your session. In this case, the plugin seems to be loaded again and shows an updated UI (if ti was to be updated).
- The plugin has very limited access to anything outside of its bundle. Finding a safe place to store your password is not easy.
- Never think that because it worked a few times, it will work the next. Stay focus and keep that SSH connection.

Useful Stuff

Three Shell scripts which allow you to install or uninstall the original login system by simply running *sh [filename].sh*.

- install.sh - Install your Plugin
- uninstall.sh - Uninstall your Plugin
- suspend.sh - Suspend the Mac

Note: For the first and the second, you will need to update the location and the name of the build.

Command to show the *SFAuthorizationPluginView* without logging out :

/System/Library/CoreServices/Menu\ Extras/User.menu/Contents/Resources/CGSession -suspend

Restoring the original login system :

security authorizationdb write system.login.console < system.login.console.original.txt

sudo killall SecurityAgent

sudo killall securityd

Wrapping Up

I hope everything went well, if you have any questions or suggestions, feel free to send me an email !