ESSO Configuring Rules for Single Sign On

Configuring Rules for Single Sign On

Configuring Single Sign-on

- User interface pattern recognition
- Web interfaces pattern recognition
- Configuring rules for basic / kerberos authentication

SSO system is configured based on the detection of administrator defined User Interface patterns. The system currently supports native Windows applications, Java applications and Web applications.

The UI Patterns are expressed with XML files associated with each application entry point. They are composed of:

- **Rules** for detecting user interfaces (defined like application attributes or elements).
- **Action** to be taken on user interface recognition. (defined with the action element for the application).

Complementary to the rules defined in Sofifd Console, the synchronization server manages a repository of user accounts and passwords, as well as other information generically known as **secrets**. In general, the system will handle any number of secrets as well as any number of accounts for each managed systems. Anyway, each account for a managed system will have only one password.

All secrets can be used and manipulated using a scripting language fully compatible with ECMA-Script, also known as Javascript.

User interface pattern recognition

The user interface detection for Windows and Java applications is done using the **Application** tag. This tag will contain one or more Component tagged elements. Each component could have many nested components. Each component could have one or more actions to perform when the user focus is at a selected component.

Next is a sample to inject the secret name “JconsolePassword” into jconsole application:

<Mazinger> <**Application** cmdLine = '.*jconsole$'> <**Component** class = 'sun.tools.jconsole.JConsole' title = 'Java Monitoring & Management Console' name = 'frame0'> <**Component** class = 'sun.tools.jconsole.JConsole\$FixedJRootPane'> <**Component** class = 'javax.swing.JPanel' name = 'null.glassPane' /> <**Component** class = 'javax.swing.JLayeredPane' name = 'null.layeredPane'> <**Component** class = 'javax.swing.JPanel' name = 'null.contentPane'> .... <**Component** class = 'javax.swing.JPasswordField' text ='' ref-as = 'password'>	Patterns to be match

<**Action** type='script' event='onFocus'>
<! [CDATA [
var account = secretStore.getAccount('soffid');
var password = secretStore.getPassword('soffid', account);
debug ('user =' + account);
debug ('password =' + password);
 	
password.setText (secretStore.getSecret ('password'));
]]>
</ **Action**>
	The action you want to be executed

....	

Thus, when the system detects that the user is within a window that meets the XML specification and the password text box is the focus owner, Soffid will execute the script action that is bound. This one will show the user password in a jconsole application field.

The Application contains in the attribute cmdLine a regular expression that is matched against the process command line. In the example, SSO will only match a running program with a command line that ends with "jconsole". It won't apply to jconsole.exe or “jconsole test”.

The element **Application** accepts the following attributes:

cmdLine	Regular expression to match the command line.

The **Component** element allows the following attributes:

class	Regular expression to validate against the kind of visual component, either a Java class or a window class.
name	Regular expression to match the name of the component. Applies only to Java components.
text	Regular expression to match the content of a text component
title	Regular expression to match the title of a java component.
dlgId	Regular expression to match window ID dialog on Windows component.
optional	If the value is true, the presence of the component is not considered critical to trigger actions associated dialogue.
check	When the check attribute has the value “**partial**”, the matcher engine considers the user interface component matches the XML pattern even when it has one or more children components that are not declared at the XML pattern. If you specify the value **full** value or the attribute is missing, the component will not match the pattern if it has children are components not specified in XML descriptor. Thus, the rule will be ignored.
ref-as	Specifies a name of a ECMA-Script variable that will refer to this component.

The **Action** element accepts the following attributes:

event	Name of the event that will trigger the action. In the current version must be set to "onFocus"
type	Indicates the type of action. Can have the following values: **setText:** Assigns a text value to the owner component. **script.** Run the specified script.
text	Text to assign, for setText actions.
repeat	If set to true, the action will be executed as many times as necessary. Otherwise, it will only run once per process.
delay	Time (in seconds) that must be elapsed before the action is executed again.

Web interfaces pattern recognition

The detection is done using the element **WebApplication**. This tag is independent of the browser used, and is based solely on the content of web document. Thus, the same rule will work both on Mozilla Firefox, Google Chrome or Internet Explorer.

<Mazinger> <**WebApplication** url = 'https://www.caib.es. *' title = 'Government of the Balearic Islands'> <**Form** action = "j_security_check"> <**Input** name="j_username" ref-as="u"/> <**Input** name="j_password" type="password" ref-as="p"/> <**Input** type="Submit" ref-as="b" />	Patterns to be match

<**Action** Type='script' event='onLoad'>
<! [CDATA [
debug('User =' + secretStore.getSecret ('user'));
debug ('password =' + secretStore.getSecret ('password'));
u.setAttribute ('value' secretStore.getSecret ('user'));
p.setAttribute ('value' secretStore.getSecret ('password'));
b.click();
]]>
</**Action**>
	Action you want to be executed

</**Form**> </**WebApplication**> </Mazinger>	

Thus, when the system detects that the browser has loaded a page matching the XML specification (url, title, and components), it will run the actions that have been associated.

Mind that despite the actions being coded in Javascript, it is not the Browser javascript engine. Thus, you cannot use browser variables or functions.

The element **WebApplication** accepts the following attributes:

url	Regular expression to match the page address
title	Regular expression to match the title of the page
content	Regular expression to match the HTML content of the page

The **Form** element will search in the HTML document for a form that matches the specified attributes. It can optionally contain one or more input elements that must be present in the HTML document. It accepts the following attributes:

id	Regular expression to match the ID attribute of the element
name	Regular expression to match the element name

method	Regular expression to match the form element's method attribute.
action	Regular expression to match the form element's action attribute.
ref-as	Specifies a name of a ECMA-Script variable that will refer to this form.
optional	A value of **true** indicates that its presence is not necessary for the execution of actions.

The **Input** element will search in the HTML document for an input element that matches the specified attributes. Input elements can be located within WebApplication or Form elements. In the first case, you will find there is any input into the document. In the second case, just find the type items included in the input form found.

id	Regular expression to match the ID attribute of the element
name	Regular expression to match the element name
type	Regular expression to match the input type
value	Regular expression to match the input value.
ref-as	Specifies a name of a ECMA-Script variable that will refer to this form.
optional	A value of **true** indicates that its presence is not necessary for the execution of actions.

The **Action** element accepts the following attributes:

event	Name of the event that will trigger the action. In the current version must be set to "onFocus"
type	Indicates the type of action. Can have the following values: **setText:** Not supported **script.** Run the specified script.
repeat	If set to true, the action will be executed as many times as necessary. Otherwise, it will only run once per process.
delay	Time (in seconds) that must be elapsed before the action is executed again.

Configuring rules for basic / kerberos authentication

Some web pages are still using basic or kerberos authentication mechanisms. These mechanisms do not present a web page to be filled in by the user. Thus, the ESSO engine cannot detect it using the method described previously.

Instead, starting from Soffid ESSO version 3.0.0, there is a new tag to teach the ESSO which credentials to send in these cases. The rules will be like the next ones:

```
<Mazinger>
  <WebTransport url="https://no-soffid.bubu.lab:4443/" system="OSCM" />
  <WebTransport url="https://no-ad.bubu.lab/" system="ad" domain="AD" />
</Mazinger>
```

The tag to use is WebTransport. It has three parameters:

Attribute	Value
url	The base url to use. Include the protocol and port number when needed. Any BASIC, NTLM or Kerberos authentication requested by that server will be automatically answered with the credentials present in the password vault
system	The ESSO will send any credential that the user has in that system. Other credentials will be ignored
domain	This is an optional attribute. It's required when trying to use Kerberos or NTLM authentication if the account name does not contain the domain name part. If the account contains the domain name, this attribute should not be present.

Due to the different ways that browsers address this kind of authentication, the user interface will be displayed according to the browser settings. For instance, Edge and Internet Explorer will display a UA dialog box.

Support and configuration tools

Introduction

> KojiKabuto.exe, the main Soffid ESSO component, picks settings and rules automatically from Soffid synchronization server at login. This configuration can be updated by running the command "KojiKabuto update". Once run, new rules will apply to all new processes. Mind that application processes that where running before the update is done will still use the old rules set.

Additionally, you can drive SSO by yourself for testing purposes. Mazinger.exe is the command line version of Soffid ESSO. It accepts the following commands:

To stop SSO service:

```
mazinger stop
```

To start Mazinger services:

```
mazinger start [-trace] [-debug] [file.mzn]
```

To get a configuration file, you can download from:
https://<synchronizationserver>:760/getmazingerconfig?user = ...

The -debug switch allows Mazinger to display all the single sign on events that are produced at users applications.

The -trace switch is only inteded for debugging and support usage.

To view all the single sign on events on a running ESSO instance, you can run:

```
mazinger debug
```

To view current SSO service status, run:

```
mazinger status
```

Mazinger can also dump XML files describing the applications user interface. This XML files can be used to describe SSO rules. To dump this XML descriptors, execute:

```
mazinger spy
```

Mazinger spy and mazinger trace are very useful when you are creating a new ESSO rule in order to see what parameters, components, atributes, ... the application are using.

In order to execute this commands, you must go to the ESSO installation directory. For example, C:\Program Files\SoffidEsso\mazinger.exe trace.

ESSO Scripting Language

Visit the ESSO Scripting Language chapter.

Configuring terminal emulation SSO

Introduction

To configure SSO on terminal emulations, an HLL API bridge has been built. This bridge allows direct communication with the terminal emulator in order to create accurate SSO rules that can be triggered based on the screen display.

Next, you have a sample rule for terminal emulation SSO:

HLL API rule

```
<Mazinger>
<HllApplication>
  <Pattern row="2">.*SOFFID.*</Pattern>
  <Pattern row="23">.*ABC.*</Pattern>
  <Action type="script" event="onMatch" repeat="true" delay="1">
     account = secretStore.getAccount ("390host");
     password = secretStore.getPassword ("390host", account);
     hll.setCursorLocation (22,3);
     hll.sendText ("HELLO "+account);
     hll.setCursorLocation (23, 3);
     hll.sendText ("YOUR PASSWORD IS "+password);
     hll.sendKeys("@E");
  </Action>
</HllApplication>
</Mazinger>
```

The rule should contain one or more patterns that will be matched against the specified row. If the screen matches all the specified patterns, the action will be executed as usual.

Nevertheless, HLL applications differ in some way from other application rules as long as the HLL engine (Sewashi) must be started separately from the ESSO engine. To active the HLL rules engine, the sewashi program must be started, specifying the HLL API used to interact with the terminal emulator, and optionally, the sessions to be managed:

```
%ProgramFiles%\SoffidESSO\Sewashi.exe --dll "%ProgramFiles%\IBM\Personal
Communications\PCSHLL32.DLL" --sessions ABCDEFG
```

To stop the HLL engine, Sewashi --stop can be executed. This program can executed from Soffid login and logout scripts.