SecurOS ACS SDK

Components of the satellite integration of an ACS

External integration of the conventional Meow ACS should contain the following files placed in the SecurOS root directory:

- acs_meow.json - JSON file describing the integration structure.
- securos.meow.dbi - DBI-file with a description of the database markup for integration objects.
- securos.meow.ddi - DDI file with a list of types of integration objects, the events they send, the commands they receive, and the states of these objects.

Also, external integration should have an integration module - an executable file that implements all the logics of interaction with both SecurOS and the ACS system.

Note. It is recommended that the integration module will be launched using an External application object. Alternately the module can be run separately but it is less preferred option. Please consider that the executable by default will be started by SecurOS service and it should fit all Windows session 0 requirements (ex. NO GUI).

If any objects of the access control system should be displayed on the map, then the integration should also provide:

- Modules / map / Map.meow.ini - description of integration objects for the map editor.
- Modules / Map / scripts / meow.js - Map integration logic described in JavaScript.
- skins / _common / acs / *. png - icons of objects on the Map.

Note. The names of all files included in the integration and all types of objects related to the integration of SecurOS must be unique. I.e. names must differ from both the file names and types of objects of the SecurOS itself, as well as the file names and types of objects of other integrations.

SecurOS searches the root directory for all files of the acs _ *. Json type, for everyone it considers this description of the external integration of ACS. For each such file, a new hierarchy of objects is added to the structure of SecurOS objects on the same level as with embedded ACS integrations. For each top-level object created in SecurOS, the ACS server (acs.exe) will wait for a connection from the integration module. When connected with it, the interaction is performed according to the protocol.

The settings of the root object of external integration are similar to the settings of the root objects of the built-in integrations, except that the connection settings to the server are not a set of separate settings, but a large text field in which free-form arbitrary data is indicated. This data is passed to the integration module as is.

Number of root objects of external integrations is limited by license. Nested object types are not limited.

Contents of acs_meow.json

The structure of the integration of ACS with three types of objects "Meow Device", "Meow Sensor" and "Meow Reader" is described by the following JSON file:

acs_meow.json

```
{
 "type": "MEOW",
 "children":
 [{
  "type": "MEOW_DEVICE",
  "children":
  [{
   "type": "MEOW_SENSOR"
  }]
 },{
  "type": "MEOW_READER"
 }]
}
```

Content of securos.meow.dbi

The root type of the integration object (MEOW) should be described as follows:

securos.meow.dbi

```
[OBJ_MEOW]
id, OBJID
name, TEXT
parent_id, OBJID
flags, INTEGER
params, TEXT

[OBJ_MEOW_EVENTS]
main_id, OBJID
source, TEXT
event, TEXT
enabled, SMALLINT
markers, TEXT
```

Each nested type of integration objects (for example, MEOW_DEVICE) should be described as follows:

securos.meow.dbi

```
[OBJ_MEOW_DEVICE]
id, OBJID
name, TEXT
parent_id, OBJID
flags, INTEGER
acs_id, TEXT

[OBJ_MEOW_DEVICE_CAMS]
main_id, OBJID
cam, OBJID
```

Content of securos.meow.ddi

The DDI file for the access control integration is similar to SecurOS native files except for the following:

- The group name for the root type must be ACS. It is necessary that the root object falls into the corresponding group of objects.
- All events must be marked with the X flag (this is a special sign of the ACS events). Only such events can be received from the ACS integration module. Also, all events with this attribute are displayed on the second tab of the settings panel of the root integration object, where you can set properties.
- The third column of the [EVENT] table contains the default event properties. These properties can be changed after the object is created. Possible values are off (the event is not processed), an empty value (no property is set), or one or more of the following properties (separated by a comma if there are more than one):
  - response_required - operator response required;
  - alarm - alarm;
  - hardware_event - equipment event;
  - access_requested - access requested;
  - access_denied - access denied;
  - access_granted - access allowed.
- The root integration object must have CONNECTED and DISCONNECTED events.

Actual examples can be found in the file securos.acs_XX.ddi, used for the access control systems integrations supplied with SecurOS.

Content of Modules / map / Map.meow.ini

The description of integration objects for the map editor sets the correspondence of the object type in SecurOS and the icon used for it in the editor. It should have the following form:

Map.meow.ini

```
; path to the object's icons relative to "skins/_common".
MEOW_READER = acs/meow_reader.png
MEOW_SENSOR = acs/meow_sensor.png
```

Actual examples can be viewed in the Modules / Map / Map.ini file, which describes all objects included in SecurOS that can be added to the map.

Content of Modules / Map / scripts / meow.js

The script of integration with the Card should have the following form:

meow.js

```
// Set object parameters when it appears on the map.
function meowLoad(o)
{
 o.image = "acs/" + o.system.type + ".png" // path to the object's icons relative to "skins/_common".
 o.drag = o.system.type + ":" + o.system.id // enables drag and drop for the object.
}

// updates object parameters when it is changed.
// also update is performed whn object appears on the map.
function meowUpdate(o)
{
 o.name = o.system.name // set and update object name on the map.
 o.state = o.system.stateName // set and update object state on the map.
 o.blinking = o.system.state == "ALARMED" // makes the object blink when it is in ALARMED state.
}

// when user has enough user rights enables commands for him
// the commands available are listed in securos.meow.ddi
function meowMenuFilter(o, action)
{
    return o.system.rights >= 2;
}

meow = { load: meowLoad, update: meowUpdate, menuFilter: meowMenuFilter }

Map.registerType("MEOW_READER", meow)
Map.registerType("MEOW_SENSOR", meow)
```

You can find samples in the Modules/Map/scripts/acs.js file used for the ACS integrations supplied with SecurOS. Also other JavaScript files for the Map xan be used as an example.

Note. The script should list all the types of ACS objects which should be displayed on maps.

Protocol of interaction of integration module and the ACS server

The protocol describes the interaction of the integration module with the ACS server in the format of IIDK messages. Messages that begin with MEOW || apply to the entire ACS system as a whole, and the actual type of the root integration object will be used instead of MEOW. Messages that begin with MEOW_DEVICE|1| refer to a specific integration object, and instead of MEOW_DEVICE, the actual type of the integration object will be used as well as the object identifier instead of 1.

Note. All object identifiers transmitted between the integration module and the ACS server are not identifiers of SecurOS objects, but arbitrary data, allowing to match SecurOS objects to entities of the ACS system. Identifiers of SecurOS objects are not passed to the integration module.

Connect to server

For each external integration root object created in the system, the ACS server waits for a connection to port 960 (TCP 20960) of an IIDK client. The client should use the connection identifier consisting of the root object type and its identifier separated by a colon (for example, MEOW: 1).

When connecting to the ACS-server, a message of the following form is sent to it:

```
MEOW||SETUP|params<...>,last_event_time<...>
```

Where:

- params - is what is listed in the "Advanced Options".
- last_event_time - date and time of the last event received from the ACS system. The date and time are transmitted in UTC ISO format (yyyy-mm-ddThh:mm:ss.iiiZ). If no event was received, the parameter will be empty. It is expected that the integration will first

receive from the ACS system and transmit to SecurOS all events that came after this time until now, and then begin to transmit actual events.

Also, a message of the following form will be sent to the module for each integration object created in the system:

```
MEOW_DEVICE|1|ADD
```

Similar messages will be sent when new integration objects are added to the SecurOS configuration.

When deleting an integration object from the SecurOS configuration, a message will be sent to the module of the following form:

```
MEOW_DEVICE|1|REMOVE
```

Note. It is guaranteed that ADD messages will be sent in hierarchy order, and REMOVE - in reverse order. It is also guaranteed that the REMOVE message will be sent only if an ADD message was sent for the corresponding object.

Each time the "Additional parameters" change, a message of the following type is sent to the integration module:

```
MEOW||UPDATE|params<...>
```

Where:

- `params` – is a new value of the "Additional parameters".

Receive the configuration

ACS integration configuration can be updated manually. To update the configuration it is necessary to press "Update configuration" button in the corresponding object's settings.

ACS configuration request is the following:

```
MEOW||GET_OBJECTS
```

The response should have the following format:

```
MEOW||OBJECTS|objects<...>
```

Where:

- `objects` – is the object tree of ACS. The object tree is described by a JSON of the following format:

```
[{
 "type": "MEOW_DEVICE",
 "id": "1",
 "name": "Meow Device 1",
 "children":
 [{
  "type": "MEOW_SENSOR",
  "id": "1",
  "name": "Meow Sensor 1"
 },{
  "type": "MEOW_SENSOR",
  "id": "2",
  "name": "Meow Sensor 2"
 }]
},{
 "type": "MEOW_READER",
 "id": "1",
 "name": "Meow Reader 1"
}]
```

When adding an object into SecurOS configuration the module will receive ADD message, and when deleting object it will receive REMOVE message (see above).

Receive events

The integration should send events to SecurOS when it receives the event from ACS. SecurOS receives only events related to the objects that exist in the SecurOS configuration (which were created by ADD message and were not later deleted by REMOVE message).

ACS events can be send in the following message:

```
MEOW_DEVICE|1|EVENT|type<...>,timestamp<...>,card_number<...>,person_id<...>
```

Where:

- `type` – event type declared in securos.meow.ddi in the `[EVENT] section`.
- `timestamp` – UTC date and time of the event, which were received from the external system. ISO format is supported (`yyyy-mm-dd Thh:mm:ss.nnnZ`).
- `card_number` – access card number realted to the event (can be empty).
- `person_id` – person id related to the event (can be empty).

Receive states

The integration should send status updates to SecurOS when the state of an object changes in the ACS or when the object is created in SecurOS (message ADD received). SecurOS receives only events related to the objects that exist in the SecurOS configuration (which were created by ADD message and were not later deleted by REMOVE message).

ACS states can be send in the following message:

```
MEOW_DEVICE|1|STATE|state<...>
```

Where:

- `state` – is the current state of a device which is declared in securos.meow.ddi in `[STATE] section`. A device can have multiple states simultaneously (ex. when a door is opened and is held simultanously). In such cases multiple states are listed separated by comma (ex. `OPEN,HOLD`).

Receive personal data

SecurOS can request the integration for personal data. The integration should send personal data to SecurOS only in response for the corresponding request. If the integration fails to receive personal data then it shouldn't reply to SecurOS.

Personal data request should have the following format:

```
MEOW||GET_PERSON|id<...>
```

Where:

- `id` – is the person id.

The following response is expected from the integration:

```
MEOW||PERSON|id<...>,name<...>,company<...>,photo<...>,cards<...>
```

Where:

- `id` – is the person id.
- `name` – is the person's full name (given name, father's name, family name).
- `company` – company the person is related to.
- `photo` – person's photo in PNG or JPEG format encoded in Base64.
- `cards` – access cards data related to that person. Each card data should have the card id, issuing date and expiration date. The list should be provided as the following JSON:

```
[{
 "number": "1111111111",
 "valid_from": "YYYY-MM-DD",
 "valid_until": "YYYY-MM-DD"
},{
 "number": "2222222222",
 "valid_from": "YYYY-MM-DD",
 "valid_until": "YYYY-MM-DD"
}]
```

Send commands

When it is necessary to execute a command in the ACS when SecurOS sends the following message related to the target device:

```
MEOW_DEVICE|1|SEND_COMMAND|command<...>
```

Where:

- `command` – is the command declared in securos.meow.ddi in `[REACT]` section.

The command cannot contain any parameters. This is a limitation of current ACS module version.

ACS SDK Sample

Sample contents

1. C++ sample source code ACS_integration_sample_source.zip.
2. Integration declaration files (.dbi, .ddi, .json, .ini, .js) required for SecurOS.
3. Executable file acs_tester.exe.

How to work with acs_tester.exe

Install

1. Unpack the `ACS_integration_sample.zip`.
2. Copy files from the archive into corresponding SecurOS folders:

Files	Path relative to SecurOS root folder
securos.meow_en.ddi securos.meow.dbi asc_meow.json	.
meow_reader.png meow_sensor.png	./skins/_common/acs
Map.meow.ini	./Modules/Map
meow.js	./Modules/Map/scripts

3. Copy acs_tester.exe into SecurOS root folder.
4. Restart SecurOS after changing `securos.dbi` properties (check "ready for archiving" flag in file properties).

SecurOS Setup

1. Configure SecurOS to work with ACS according to SecurOS Administration Guide.
2. Create ACS Meow object.

Start

1. Start acs_tester.exe from command line passing ACS Meow object id as a parameter. If everything is configured properly then acs_tester will connect to ACS server and print the following in the console:

```
--> CONNECTED
--> MEOW||SETUP|params<...>,last_event_time<...>
```

2. Perform a configuration update by clicking on the "update configuration" button from the settings panel of the ACS Meow object. Two sensors and two readers will be created in SecurOS and random events and statuses will be generated using acs_tester.exe.
3. You can work with the ACS Meow demo object from the ACS operator interface, VB/Jscripts, Maps and Media Client according to the documentation.

Integration module implementation

When writing the integration module, it is necessary to implement the logic according to the protocol of interaction with the ACS server and generate all service files defining the integration structure.

Connect to server

The connection is made according to the IIDK documentation using the ConnectEx function on the "port" 960 with the connection id <type of top-level integration object>: <id in SecurOS>.

controller.cpp

```
if (0 == ConnectEx("127.0.0.1", "960", m_id.c_str(), handler, (size_t)this, 1, 1))
{
    std::cout << std::red << "Failed to connect to ACS" << std::white << std::endl;
    QCoreApplication::exit(-1);
}
```

In case of successful connection, CONNECTED, SETUP and, possibly, ADD messages will be received according to the protocol.

Interact with the configuration

When the GET_OBJECTS message is received, the integration module must respond with an OBJECTS message containing the actual configuration of integration objects in the json format in the objects parameter.

controller.cpp

```
if (mi.action == "GET_OBJECTS") // ACS server retrieves configuration
{
    send(QString("MEOW||OBJECTS|objects<%1>").arg(m_config).toStdString());
}
```

controller.cpp

```
void Controller::send(const std::string& message) const
{
    SendMsg(m_id.c_str(), message.c_str());
    std::cout << std::green << "<-- " << message << std::white << std::endl;
}
```

Work with events and states

1. When SETUP is received, if last_event_time is not empty, then it is expected that all events from the specified time point to the current time will be transmitted first, and only then actual events will be transmitted.
2. The integration module should transmit current events to the ACS server as they are received.
   acs_tester.exe generates random events for demonstration purposes.

controller.cpp

```
void Controller::sendEvent() // Send event to ACS server
{
    // Some EVENTs of MEOW_READER from *.ddi file
    static const QStringList events{"LOCK_MONITOR_OPEN", "LOCK_MONITOR_SECURE",
"EXIT_OPEN_WITH_TAMPER", "EXIT_CLOSED_WITH_TAMPER", "DOOR_OPEN",
"DOOR_FORCED", "DOOR_CLOSED"};

    const auto& readers = m_devices.values("MEOW_READER");
    if (readers.isEmpty())
        return;

    const auto& person = m_persons.value(m_persons.keys().value(std::uniform_int_distribution<>(0,
m_persons.keys().size() - 1)(m_random)));
    const auto& cards = person.value("cards").toArray();


send(QString("MEOW_READER|%1|EVENT|type<%2>,timestamp<%3>,card_number<%4>,person_id<
%5>").arg(
        readers.value(std::uniform_int_distribution<>(0, readers.size() - 1)(m_random)),
        events.value(std::uniform_int_distribution<>(0, events.size()-1)(m_random)),
        QDateTime::currentDateTimeUtc().toString(Qt::ISODateWithMs),
        cards.at(std::uniform_int_distribution<>(0, cards.size() -
1)(m_random)).toObject()["number"].toString(),
        person["id"].toString()).toStdString());
}
```

3. The integration module should transmit the current state of the objects to the ACS server as soon as it received the ADD message, or after updating the state of the object on the ACS side.
   acs_tester.exe generates random object states for demonstration purposes.

controller.cpp

```
void Controller::sendState()  // Send device state to ACS server
{
    // Some STATEs of MEOW_READER from *.ddi file
    static const QStringList states{ "CARD_ONLY", "DISABLED", "ENABLED" };

    const auto& readers = m_devices.values("MEOW_READER");
    if (readers.isEmpty())
        return;

    send(QString("MEOW_READER|%1|STATE|state<%2>").arg(
        readers.value(std::uniform_int_distribution<>(0, readers.size() - 1)(m_random)),
        states.value(std::uniform_int_distribution<>(0, states.size() - 1)(m_random))).toStdString());
}
```

Work with personal data

For each GET_PERSON message the integration module should respond with a PERSON message if personal data corresponding to the specified identifier is found.

controller.cpp

```
else if (mi.action == "GET_PERSON") // ACS server retrieves personal information
{
    const QString& id = mi.params.value("id");
    const auto& it = m_persons.find(id);
    if (it == m_persons.end())
    {
        std::cout << std::red << "Unknown person id=" << id.toStdString() << std::white << std::endl;
        return;
    }


     QStringList params;
     for (const auto& key : it.value().keys())
          params << QString("%1<%2>").arg(key, key == "cards" ? QJsonDocument{ it.value().value(key).toArray()
}.toJson(QJsonDocument::Compact) : it.value().value(key).toString());

     send(QString("MEOW||PERSON|%1").arg(params.join(',')).toStdString());
}
```

Work with commands

Upon receiving the SEND_COMMAND message, the integration module should execute the command from the command parameter on the ACS side.

acs_tester.exe for demonstration purposes simply displays the command to the console.

controller.cpp

```
else if (mi.action == "SEND_COMMAND") // ACS server sends command to us
{
    std::cout << std::blue << "Command " << mi.params.value("command").toStdString() << " for " <<
mi.source.toStdString() << ":" << mi.id.toStdString() << std::white << std::endl;
}
```

Files

Integration sample

ACS_integration_sample.zip

Integration sample C++ source code

ACS_integration_sample_source.zip

C# wrapper for iidk.dll

ISS.SecurOS.idk.dll