Desarrollo en Medulla

- Cómo ampliar Medulla
- Cómo utilizar la API XML-RPC de Medulla

Cómo ampliar Medulla

En primer lugar, hay algunas cosas importantes que hay que tener en cuenta al desarrollar en torno al agente:

El agente de la máquina se actualiza automáticamente cuando el código no es idéntico entre el agente y el código base del agente que se encuentra en el servidor. Para desactivar este comportamiento, es necesario añadir la siguiente configuración a agentconf.ini y reiniciar el agente:

```
[updateagent]
updating = 0
```

Los complementos también se actualizan automáticamente si ha cambiado la versión. Si es necesario modificar un complemento, no actualice la versión hasta que se hayan realizado todas las pruebas.

Hay cuatro tipos de complementos para el agente de máquina: complementos de inicio, complementos de actualización, complementos de acción y complementos programados.

1. Los complementos de inicio son aquellos que se ejecutan cuando se inicia el agente y se definen en start_machine.ini;
2. Los complementos de actualización se utilizan para instalar o actualizar componentes externos utilizados por el agente;
3. Los complementos de acción son invocados por una acción recibida por el agente;
4. Los complementos programados son aquellos que se invocan a una hora o intervalo específicos.

Cada plugin puede tener su propio archivo de configuración con el mismo nombre que el plugin y debe añadirse al siguiente parámetro en agentconf.ini para que se cargue la configuración:

```
[plugin]
pluginlist = xxxxxxx, yyyyyyy
```

Los complementos programados tienen su propia programación definida en el complemento, en el parámetro SCHEDULE. Sin embargo, esto se puede anular en el archivo manage_scheduler_machine.ini

---

Hay tres formas de ampliar Medulla:

1. interacción con el agente de la máquina a través de un socket TCP
2. interacción con el agente de la máquina a través de tuberías con nombre

3. complementos de acción del agente de máquina
4. complementos programados del agente de máquina

Cómo interactuar con el agente a través de un socket TCP

Es necesario definir una nueva acción en**server_kiosk.py**, en la función denominada **handle_client_connection**, en la condición

```
if 'action' in result:
```

y esta acción debe añadirse al mensaje JSON que se envía a través del socket TCP.
A continuación se muestra un ejemplo del mensaje enviado:

```
{
    "action": "myNewAction",
    "sessionid": "mysessionid",
    "base64": false,
    "data": {
        "date": "2020-06-24T15:45:02.000Z",
        "family1": {
            "field1": "value1",
            "field2": "value2"
        },
        "family2": {
            "field1": "value1",
            "field2": "value2"
        }
    }
}
```

El contenido anterior debe guardarse en un archivo llamado**json_file**para enviarlo a través de un socket TCP o añadirse a una variable llamada**json_message**en tu código para enviarlo a través de tuberías con nombre

Y su equivalente en la función**handle_client_connection**de la clase**manage_kiosk_message**.
En la siguiente sección:

```
try:
    _result = json.loads(minifyjsonstringrecv(msg))
```

Añade

```
if _result['action'] == "myNewAction":
    substitute_recv = self.objectxmpp.sub_monitoring
    logging.getLogger().warning("send to %s to %s" % (_result,substitute_recv ))
    self.objectxmpp.send_message(mbody=json.dumps(_result),
                                 mto=substitute_recv,
                                 mtype='chat')
    return
```

Y en el siguiente bucle:

```
if 'action' in result:
    if result['action'] == "kioskinterface":
...
```

Añadir

```
elif result['action'] == "myNewAction":
    datasend['action'] = "myNewSubstituteAction"
    subs_recv = self.objectxmpp.sub_monitoring
    datasend['sessionid'] = getRandomName(6, "mynewsubstituteaction")
    datasend['data'] = result['data']
```

El ejemplo anterior enviará a su vez el mensaje al JID de sub_monitoring con una nueva acción que debe llevarse a cabo: myNewSubstituteAction

El puerto TCP de escucha se define en**el parámetrokiosk/am_local_portdel archivo agentconf.ini**. El valor predeterminado es 8765.

A continuación se muestra un ejemplo de emisor TCP escrito en Python para el siguiente paso, que es el envío real de los datos al socket TCP:

```
#!/usr/bin/env python
# -*- coding: utf-8; -*-
#
# (c) 2023 siveo, http://www.siveo.net
#
# Este archivo forma parte de Medulla, http://www.siveo.net
#
# Pulse 2 es software libre; puede redistribuirlo y/o modificarlo
# bajo los términos de la Licencia Pública General de GNU tal y como la publica
# la Free Software Foundation; ya sea la versión 2 de la Licencia, o
# (a su elección) cualquier versión posterior.
#
# Pulse 2 se distribuye con la esperanza de que sea útil,
# pero SIN NINGUNA GARANTÍA; ni siquiera la garantía implícita de
# COMERCIABILIDAD o IDONEIDAD PARA UN FIN DETERMINADO.  Consulte la
# Licencia Pública General de GNU para obtener más detalles.
#
# Debería haber recibido una copia de la Licencia Pública General de GNU
# junto con Medulla; si no es así, escriba a la Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, EE. UU.
# archivo clientTCPcli.py

# Ejecute python clientTCPcli.py  -p ./file.json en la máquina cliente para
# inyectar los datos

from optparse import OptionParser

import socket
import sys
```

```
import os
import select
def send_message(message, host, port, timeout_in_seconds = 5):

    # Crear un socket TCP/IP
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Conectar el socket al puerto en el que el servidor está a la escucha


    try:

        server_address = (host, port)
        print >>sys.stderr, 'conectando a %s puerto %s' % server_address
        sock.connect(server_address)
    except socket.error , msgerror:
        print 'Error al vincular en la interfaz de comando ' + host + ' puerto ' + str(port) + '
        sys.exit(str(msgerror[0]))
    try:
        # Enviar datos
        print >>sys.stderr, 'enviando "%s"' % mensaje
        sock.sendall(mensaje)

        # Buscar la respuesta
        cantidad_recibida = 0
        cantidad_esperada = len(mensaje)
        ready = select.select([sock], [], [], timeout_in_seconds)
        data = ""
        if ready[0]:
            data = sock.recv(4096)
            return 0, data
        return -1,""
    finally:
        print >>sys.stderr, 'cerrando socket'
        sock.close()


if __name__ == '__main__':
    optp = OptionParser()
    #optp.add_option("-h", "--help",action="store_true",
                        #dest="help", default=False,
                                #help="host")

    optp.add_option("-H", "--host", action="store_true",
                        dest="host", default="localhost",
                                help="host")

    optp.add_option("-P", "--port",
                        dest="port", default=8765,
                        help="port connection")

    optp.add_option("-T", "--timeout",
                        dest="timeout_in_seconds", default=2,
                            help="Tiempo de espera de recepción en segundos")
```

```
    optp.add_option("-m", "--msg",
                        dest="msg", default = "",
                            help="mensaje para enviar al servidor TCP")

    optp.add_option("-p", "--pathfile",
                        dest="pathfile", default = None,
                            help="archivo de contenido para enviar al servidor TCP")

    opts, args = optp.parse_args()
    #if opts.help:
        #print "uso del comando"
        #os._exit(0)
    message = ""
    if opts.pathfile is None:
        if opts.msg =="":
            print "opción faltante < -m | -p> "
            sys.exit(-1)
        message = opts.msg
    else:
       if os.path.exists(opts.pathfile):
            with open(opts.pathfile, 'r') as f:
                message =  f.read()

    if message !="":
        code, msg = send_message(message, opts.host, opts.port, opts.timeout_in_seconds)
    print "código de error %s, respuesta del servidor %s"%(code,msg)
```

Para enviar el mensaje:

```
python clientTCPcli.py -p <archivo_json>
```

Cómo interactuar con el agente a través de tuberías con nombre

La forma de interactuar con el agente a través de tuberías con nombre es la misma que para interactuar con el agente a través de un socket TCP, excepto en lo que respecta al envío del mensaje.

Para enviar el mensaje a la tubería con nombre, aquí hay un código de ejemplo escrito en Python:

```
import win32file

def send_message(json_message):
    fileHandle = win32file.CreateFile("\\\\.\\pipe\\interfacechang",
                                      win32file.GENERIC_READ | win32file.GENERIC_WRITE,
                                      0,
                                      None,
                                      win32file.OPEN_EXISTING,
                                      0,
```

```
                                    None)
    win32file.WriteFile(fileHandle, json_message)
    win32file.CloseHandle(fileHandle)
```

Cómo escribir complementos de acción para el agente

A continuación se muestra una plantilla que se puede utilizar para escribir complementos de acción:

```
import logging
import json

plugin = {"VERSION": "1.0", "NAME": "mynewaction",  "TYPE": "machine"}

logger = logging.getLogger()

def action( objectxmpp, action, sessionid, data, message, dataerror):
    logger.debug("###################################################")
    logger.debug("llamada %s desde %s id de sesión %s" % (plugin, message['from'], sessionid))
    logger.debug("###################################################")
    datasend = {"action" : "myNewSubstituteAction",
                "data" : data,
                "sessionid": sessionid,
                "ret": 0,
                "base64": False
                }
    objectxmpp.send_message(mto=objectxmpp.sub_monitoring,
                            mbody=json.dumps(datasend),
                            mtype='chat')
```

Ten en cuenta lo siguiente:

- NAME debe coincidir con el nombre del archivo del complemento. En este caso, el archivo se llamará plugin_mynewaction.py
- TYPE debe definirse como machine, relayserver o all, dependiendo de su destino
- La función de acción será el código que se ejecutará por defecto. El ejemplo anterior enviará a su vez el mensaje al JID sub_monitoring con una nueva acción que se llevará a cabo: myNewSubstituteAction

---

Cómo escribir complementos programados para el agente

A continuación se muestra una plantilla que se puede utilizar para escribir complementos programados:

```
import logging
import json
import os
import ConfigParser
from pulse_xmpp_agent.lib.agentconffile import directoryconffile
from pulse_xmpp_agent.lib.utils import file_put_contents

plugin = {"VERSION": "1.0", "NAME": "scheduling_mynewscheduledaction", "TYPE": "machine", "SCHED

SCHEDULE = {"schedule" : "*/15 * * * *", "nb" : -1}

logger = logging.getLogger()

def schedule_main(xmppobject):
    logger.debug("=====================================================")
    logger.debug(plugin)
    logger.debug("=====================================================")
    if xmppobject.num_call_scheduling_mynewscheduledaction == 0:
        __read_conf(xmppobject)

    if xmppobject.config.mynewscheduledaction_enable:
        data = {}
        data['family1'] = {}
        data['family1']['field1'] = "value1"
        data['family1']['field2'] = "value2"
        data['family2'] = {}
        data['family2']['field1'] = "value1"
        data['family2']['field2'] = "value2"

    if xmppobject.config.mynewscheduledaction_forward:
        datasend = {"action" : "myNewSubstituteAction",
                    "data" : data,
                    "sessionid": "mysessionid",
                    "base64": False
                    }
        objectxmpp.send_message(mto=objectxmpp.sub_monitoring,
                                mbody=json.dumps(datasend),
                                mtype='chat')

def __read_conf(xmppobject):
    """
        Lee la configuración del complemento
    """
    configfilename = os.path.join(directoryconffile(), "%s.ini" % plugin['NAME'])
    logger.debug("Leyendo la configuración en el archivo %s" % configfilename)

    #parámetros predeterminados
    xmppobject.config.mynewscheduledaction_enable = True
    xmppobject.config.mynewscheduledaction_forward = False

    if not os.path.isfile(configfilename):
        logger.warning("Falta el archivo de configuración %s del complemento %s" % (plugin['NAME
        logger.warning("El archivo de configuración que falta se creará automáticamente.")
        file_put_contents(configfilename,
```

```
                        "[mynewscheduledaction]\n" \
                        "enable = 1\n" \
                        "forward = 0\n")

    # Cargar la configuración desde el archivo
    Config = ConfigParser.ConfigParser()
    Config.read(nombre_archivo_conf)
    if os.path.exists(nombre_archivo_conf + ".local"):
        Config.read(nombre_archivo_conf + ".local")
    if Config.has_section("mynewscheduledaction"):
        if Config.has_option("mynewscheduledaction", "enable"):
            xmppobject.config.mynewscheduledaction_enable = Config.getboolean('mynewscheduledact
        if Config.has_option("mynewscheduledaction", "forward"):
            xmppobject.config.mynewscheduledaction_forward = Config.getboolean('mynewscheduleda
```

Ten en cuenta lo siguiente:

- NAME debe coincidir con el nombre del archivo del complemento. En este caso, el archivo se llamará plugin_mynewaction.py
- TYPE debe definirse como machine, relayserver o all, dependiendo de su destino
- La notación SCHEDULE es similar a la notación cron. El parámetro adicional nb define cuántas veces debe ejecutarse el complemento. Si es -1, se ejecutará indefinidamente
- La función schedule_main será el código que se ejecutará por defecto. El ejemplo anterior leerá un archivo de configuración o lo creará si no existe, y enviará un mensaje al JID sub_monitoring con una nueva acción que se debe llevar a cabo: myNewSubstituteAction

Cómo utilizar la API XML-RPC de Medulla

Esto guiará al usuario en el uso de la API XML-RPC de Medulla: Medulla's XML-RPC API.php

1: Configuración

Compruebe lo siguiente:

- Certificado SSL: Obtenga el certificado del servidor de Medulla:
  http://<medulla_server>/downloads/medulla-ca-chain.cert.pem
- Instala el certificado en el servidor que va a consultar la API
  - Para sistemas basados en Debian, copie el archivo a*/usr/local/share/ca-certificates*y cambie su extensión a crt; a continuación, importe el certificado ejecutando el comando*update-ca-certificates*
  - Para sistemas basados en RedHat, copie el archivo a*/etc/pki/ca-trust/source/anchors* y cambie su extensión a crt; a continuación, importe el certificado ejecutando el comando*update-ca-trust extract*
- Si es necesario, añada la IP y el nombre de host del servidor XML-RPC a /etc/hosts

2: Uso de la API

Primero autentique al usuario y obtenga la cookie de sesión:

```
/**
 * Ejecuta una solicitud XML-RPC.
 *
 * @param string $method El nombre del método XML-RPC que se va a llamar.
 * @param array $params Los parámetros que se van a pasar al método.
 * @param bool $includeCookie Indica si la cookie de sesión debe incluirse en la solicitud.
 * @return array Un array que contiene los encabezados HTTP y el cuerpo de la respuesta.
 * @throws Exception Si se produce un error al conectarse o al enviar la solicitud.
 */
function executeRequest($method, $params, $includeCookie = false) {
    $agentInfo = $_SESSION["XMLRPC_agent"];
    $requestXml = xmlrpc_encode_request($method, $params, ['output_type' => 'php', 'verbosity' =

    // Definimos los encabezados HTTP
    $url = "/";
    $httpQuery  = "POST " . $url . " HTTP/1.0\r\n";
    $httpQuery .= "User-Agent: MMC web interface\r\n";
    $httpQuery .= "Host: " . $agentInfo["host"] . ":" . $agentInfo["port"] . "\r\n";
    $httpQuery .= "Content-Type: text/xml\r\n";
```

```
    $httpQuery .= "Content-Length: " . strlen($requestXml) . "\r\n";
    // Se añade la cookie si es necesario
    if ($includeCookie) {
        $httpQuery .= "Cookie: " . $_SESSION['cookie'] . "\r\n";
    }
    $httpQuery .= "Authorization: Basic " . base64_encode($agentInfo["login"] . ":" . $agentInfo
    $httpQuery .= $requestXml;

    // Configurar el contexto SSL
    // 'allow_self_signed' se establece en false para aceptar solo certificados firmados por una
    // 'verify_peer' se establece en true para verificar el certificado SSL del servidor
    $context = stream_context_create();
    $proto = $agentInfo["scheme"] == "https" ? "ssl://" : "";
    if ($proto) {
        stream_context_set_option($context, "ssl", "allow_self_signed", false);
        stream_context_set_option($context, "ssl", "verify_peer", true);
    }

    // Abrimos la conexión con el servidor
    $socket = stream_socket_client($proto . $agentInfo["host"] . ":" . $agentInfo["port"], $errn
    if (!$socket) {
        throw new Exception("No se puede conectar al servidor XML-RPC: $errstr ($errno)");
    }

    // Se añade un tiempo de espera de 60 segundos para la lectura
    stream_set_timeout($socket, 60);

    if (!fwrite($socket, $httpQuery)) {
        throw new Exception("No se pueden enviar datos al servidor XML-RPC");
    }

    $responseXml = '';
    while (!feof($socket)) {
        $ret = fgets($socket, 128);
        $responseXml .= $ret;
    }
    fclose($socket);

    // Separa los encabezados HTTP del cuerpo de la respuesta
    list($headers, $body) = explode("\r\n\r\n", $responseXml, 2);
    return [$headers, $body];
}

/**
 * Autentifica al usuario y recupera la cookie de sesión.
 * @param string $method El nombre del método XML-RPC que se va a llamar.
 * @param array $params Los parámetros que se van a pasar al método.
 * @return string La cookie de sesión.
 * @throws Exception Si se produce un error durante la autenticación.
 */
function authenticateAndGetCookie($method, $params) {
    list($headers, $body) = executeRequest($method, $params);
```

```
    // Análisis de los encabezados HTTP para extraer la cookie de sesión
    $headers_array = array();
    $header_lines = explode("\r\n", $headers);
    foreach ($header_lines as $header) {
        $parts = explode(': ', $header, 2);
        if (count($parts) == 2) {
            $headers_array[$parts[0]] = $parts[1];
        }
    }

    // Uso de los encabezados analizados
    if (isset($headers_array['Set-Cookie'])) {
        $cookie = $headers_array['Set-Cookie'];
        $_SESSION['cookie'] = $cookie;
    } else {
        throw new Exception('Autenticación fallida, no se ha recibido ninguna cookie');
    }

    return $cookie;
}
```

3: Ejemplos de solicitudes

- Obtener la lista de todos los paquetes

```
/**
 * Envía una solicitud XML-RPC y devuelve la respuesta solo si se ha autenticado mediante cookie
 * @param string $method El nombre del método XML-RPC que se va a llamar.
 * @param array $params Los parámetros que se van a pasar al método.
 * @return array La respuesta XML-RPC.
 * @throws Exception Si se produce un error al enviar la solicitud.
 */
function sendXmlRpcRequest($method, $params) {
    list($headers, $body) = executeRequest($method, $params, true);
    $responseXml = substr($body, strpos($body, '<?xml'));
    $response = xmlrpc_decode($responseXml, 'UTF-8');
    if (is_array($response) && xmlrpc_is_fault($response)) {
        throw new Exception
("Error XML-RPC: {$response['faultString']} ({$response['faultCode']})");
    }
    return $response;
}

    $method = "pkgs.get_all_packages";
    $params = [
        'root', // inicio de sesión
        false,  // sharing_activated
        0,      // inicio
        10,      // fin
        [
            'filter' => 'hostname', // nombre de ejemplo del paquete
```

```
        ] // ctx
    ];

    try {
        $responseXml = sendXmlRpcRequest($method, $params);

    } catch (Exception $e) {
        echo "Error: " . $e->getMessage();
    }
```

Consejos: Para obtener el UUID del paquete: $respuestaXml['datas']['uuid'][$key]

- Obtener la lista de todas las máquinas

```
$method = "xmppmaster.get_machines_list";
$params = [
     0, // inicio
     20, // fin
     [
        'filter' => '',
        'field' => 'allchamp',
        'computerpresence' => 'presence', // Definir si se quieren las máquinas presentes
        // 'computerpresence' => 'no_presence', // Definir si se quieren las máquinas no present
        'location' => "UUID0", // glpi_id - entidad
    ] // ctx
];

try {
    $responseXml = sendXmlRpcRequest($method, $params);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
```

- Obtener los detalles de una máquina

```
$method = "glpi.getLastMachineInventoryPart";
$params = [
     $uuid,
     'Summary',
      0, // minbound
      0, // maxbound
      "", // filter
      [
           "hide_win_updates" => false,
           "history_delta" => false
       ], // opciones
];

try {
    $responseXml = sendXmlRpcRequest($method, $params);
```

```
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
```

Consejos: Para obtener el UUID de la máquina: $reponseXml['data']['uuid_inventorymachine'][$key]. Esto también se corresponde con el ID de máquina de GLPI precedido por el UUID

- Implementar un paquete específico en una máquina de destino

```
$method = "msc.add_command_api";
$pid = "96982fce-hostname_vq918j7wtnwzm610by"; // pid - id_del_paquete
$target = "UUID1"; // target - uuid_de_la_máquina

$params = [
    $pid,  // pid - id_del_paquete
    $target,  // target - uuid_de_la_máquina
    array(  // parámetros
    "name" => "devdemo-win-1",
    "hostname" => "devdemo-win-1",
    "uuid" => $target,
    "gid" => NULL,
    "from" => "base|computers|msctabs|tablogs",
    "pid" => $pid,
    // "title" => "TÍTULO DE LA IMPLEMENTACIÓN",
    "create_directory" => "on",
    "start_script" => "on",
    "clean_on_success" => "on",
    "do_reboot" => "",
    "do_wol" => "",
    "do_inventory" => "on",
    "next_connection_delay" => "60",
    "max_connection_attempt" => "3",
    "maxbw" => "0",
    "deployment_intervals" => "",
    "tab" => "tablaunch",
    "issue_halt_to" => array(),
    ),
    "push",  // modo
    NULL,  // gid
    array(),  // proxy
    0  // cmd_type
];

    try {
        $responseXml = sendXmlRpcRequest($method, $params);

        $commandId = $responseXml; // Recuperar el ID del comando

        $method2 = "xmppmaster.addlogincommand";
        $params2 = [
            'root', // inicio de sesión
```

```
            $commandId, // ID del comando
            '', // ID del grupo
            '', // n.º de máquinas en el grupo
            '', // número de máquinas para ejecución
            '', // fecha y hora de ejecución
            '', // paquete de parámetros
            0, // reinicio requerido
            0, // apagado requerido
            0, // ancho de banda
            0, // sincronización
            [] // parámetros
        ];

        try {
            $responseXml2 = sendXmlRpcRequest($method2, $params2);

        } catch (Exception $e) {
            echo "Error: " . $e->getMessage();
        }

    } catch (Exception $e) {
        echo "Error: " . $e->getMessage();
}
```

- Obtener registros de implementación (Audit) de sessionname

```
$method = "xmppmaster.getlinelogssession";
$params = [
    "command63bb5ee8fc834eae89" // nombre de sesión
];

try {
    $responseXml = sendXmlRpcRequest($method, $params);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
```

Consejos: Para obtener el nombre de sesión de la implementación: $reponseXml['tabdeploy'] ['sessionid'][$key]

- Obtener todos los registros de implementación por usuario y/o destino

```
$method = "xmppmaster.get_deploy_by_user_with_interval";
$params = [
    "root",       // usuario de inicio de sesión
    "",           // estado de implementación
    86400,        // intervalo de búsqueda
    0,            // inicio de paginación
    "20",         // fin de paginación
    "spo-win-1",  // filtro - nombre_de_host_de_la_máquina
    "command"     // tipo_de_implementación
```

```
];

try {
    $responseXml = sendXmlRpcRequest($method, $params);
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
```

Nuevo impulso