TP4 - IT Cybersecurity - Attack script

Operational objectives :

- Understand the purpose of executing a Python script (in a command prompt)
- To be able to define and implement a filtering rule on the IP address of the Client computer
- Be able to define and implement a filtering rule on the IP/MAC address of the Client computer
- Be able to demonstrate the effectiveness of this type of filtering

Prerequisites :

- Master the basic operations of Control Expert
- Be able to launch Vijeo Designer in simulation mode
- Understand the basic structure of a Modbus frame and its main functions
- Have understood the main principles of SNi40 configuration

The problem posed:

Rejecting (potentially hostile) Modbus requests from a station with an IP/MAC address that is not legitimized.

Resources :

- Manufacturer documentation
  - Schneider Electric
    - ➢ website
    - ➢ protocol-modbus.pdf
  - Stormshield :
    - ➢ SNS - User and Configuration Manual

- **Specific documentation**
  - Architectures Maquette Cybersec_anglais.pptx

- **Applications made available for the realization of this TP :**
  - M580 application (Control Expert): md1ae58ecyb.stu
  - HMI application (Vijeo Designer):MD1AE58ECYB
  - Default SNi40 Firewall configuration file (SNI40-TP2-0.na)
  - Attack program (Modbus) written in Python (V2.7) : Attack_Modbus.py
    (to be launched in the command prompt window)
- **Software provided, to be installed on the work PC (console) for the realization of this TP:**
  - Control Expert (Schneider Electric) : Programming of Schneider Electric M340, M580, ...
  - Vijeo Designer V6.2 SP8 : Design of Magelis HMI applications (execution including in Simulation mode on the Workstation)
  - Web Gate Client (Schneider Electric): complement to Vijeo Designer [option] (remote client of the Magelis HMI, running in an Internet Browser)
  - Internet Explorer : Microsoft Internet Browser
  - Angry IP Scanner (angryip.org): check for accessible IP addresses in a given range [option]
  - Wireshark (Wireshark Foundation): observation of Ethernet frame details
  - WinSCP/Putty: manipulation of Custom Patterns files on SNi40
  - Python code execution environment (V2.7.15)

Evaluation criteria :	😀	😐	😟
Execution and interpretation of a Modbus/TCP Attack Script (Python)			
on and implementation of an IP address filtering rule			
Definition and implementation of a filtering rule on IP/MAC address			
Demonstration of the effectiveness of this filtering by blocking			
Autonomy - Quality of work/restitution			

Time spent :	1 h	**Objective(s) :**		Comment(s) :
Evaluation :	/ 20	Reached(s)	Not reached	

TP4 - Control Lockout Protection

A **Python** script is executed on a Windows computer connected to the test platform network This script allows the speed of the motorized drive to be varied randomly (faster, slower)

We will first be asked to observe the behaviour of the motorized drive when this script is executed, and then to set up a configuration on the SNi40 firewall that allows :

- avoid variations in the speed setpoint of the motorized variable speed drive
- prevent the motorized drive from stopping

- Behaviour of the Motor-Variator

  We will check that when the attack script is launched, it varies the Speed setpoint in a pseudo-random way every 5 seconds. Then, after 30 seconds, this Python script stops the fan.

- As the attack is carried out from a PC whose IP is supposed to be distinct from that of the control PC, one could establish a rule authorizing as flows arriving on the ePAC only the flows coming from the IP address of this control PC, for example, (this being valid also for write operations on the ePAC), and excluding all other source IP addresses.

  In other words, the first objective is to set up a specific IP filtering (that of the Control PC), in order to admit only this one, and thus prevent the Windows client executing (on another PC) the Python script from communicating with the PLC, and thus from varying the Speed setpoint while stopping the engine.

- We can then try to create an object representing the Control Station, specifying its MAC address, and designate this object as the source of the rule; so that in the event that a third party station tries to usurp the IP address of this Control Station, it will not be able to cross the SNi40 firewall.
  - If alarms are raised, they should result in blocking behaviour and not just analysis. In essence, this will be done by :
  - The source object allowed in the filter rule (linked to the IP / MAC address of the Control PC)
  - The IPS profile of the industrial protocol (Modbus) already detailed in the previous tutorials, now blocking (if necessary implicitly) intrusive flows from third party machines.

Remarks

This test sequence will be played by default according to the so-called Phase 2 architecture, i.e. integrating the SNi40 firewall, which is connected to an Ethernet port (DIO) of the M580 CPU.

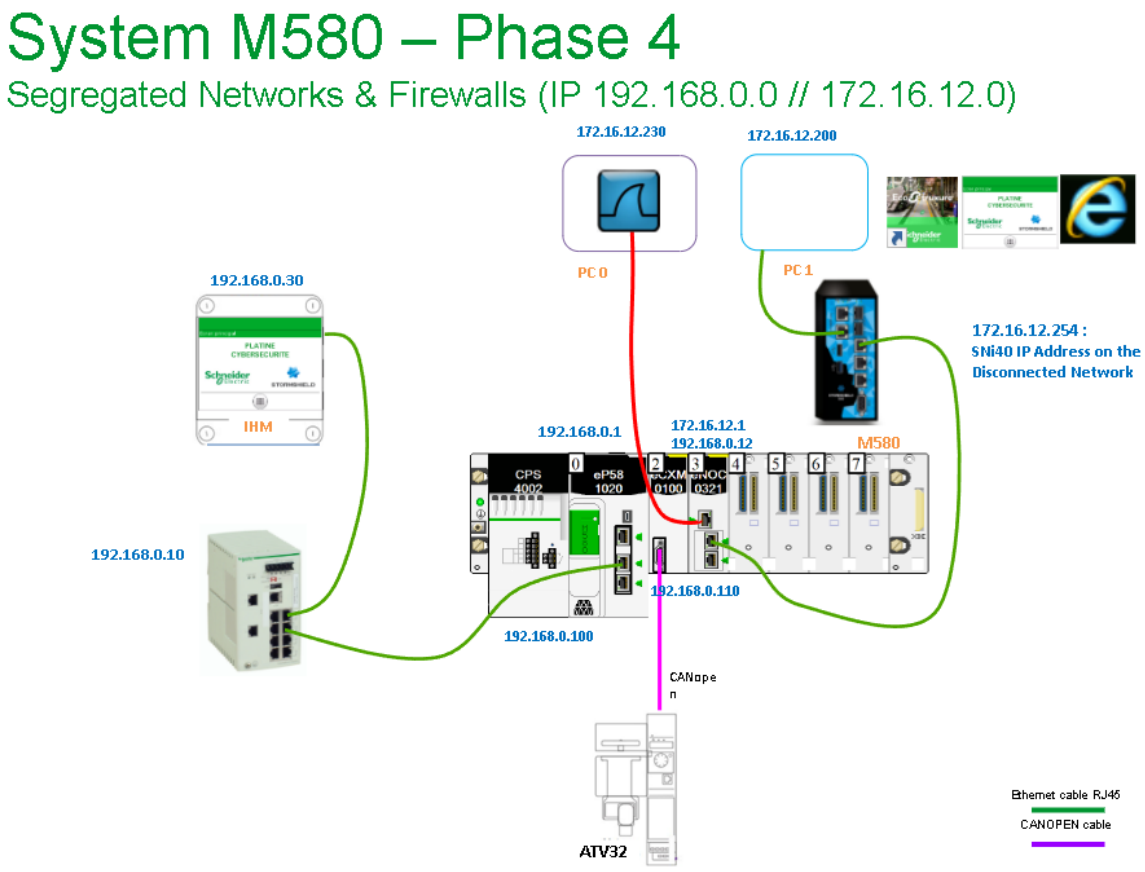
It could just as easily be replayed according to the so-called Phase 4 architecture, i.e. the one involving a separation of networks: 'Control Network' vs 'Devices Network'. This time, the SNi40 firewall is not connected to an Ethernet DIO port on the CPU module, but to an Ethernet DIO port on the BME NOC 0321 module.

With this Phase 4 architecture, the elements present on the 'Control' network will be registered in the **172.16.12.0** addressing domain, while those present on the 'Equipment' network will remain in the **192.168.0.0** addressing domain (i.e. the same as before).

We will adapt the properties of the Machine Object accordingly (Cf §2)

--- See TP#2, Chapters 11 & 12, for wiring and initializing the routing on the Workstation —

(photo below)



—

Details of expected operations

1. **Behaviour of the Drive**

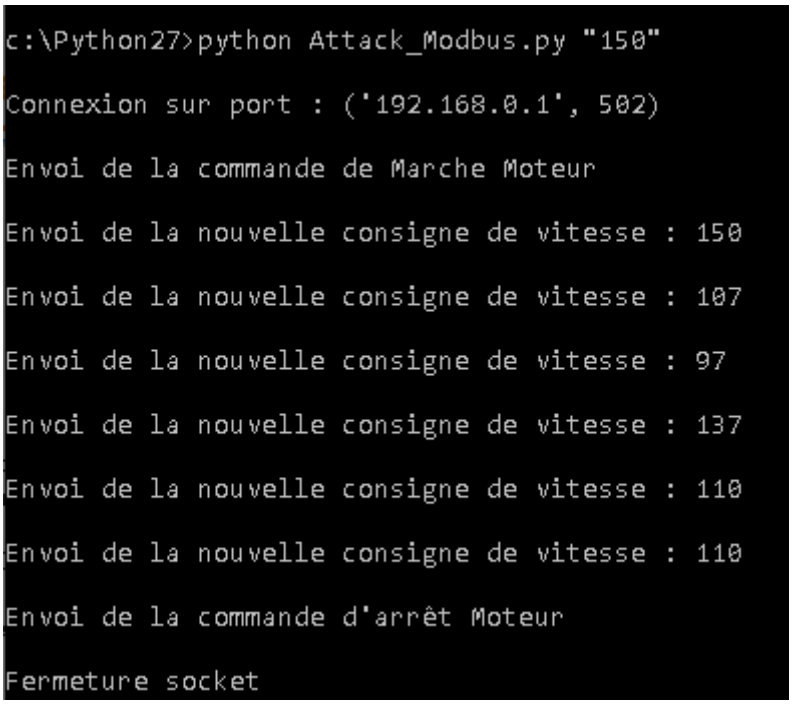
   We check that when the attack script is launched, it varies the Speed setpoint in a pseudo-random way every 5 seconds. Then, after 30 seconds, this Python script stops the fan.

At its simplest :

- Copy the attack script to C :
- After ensuring that Python (V2.7) is installed on the machine used for the attack, open a command prompt
- Reference yourself to the root of C :
- Start the attack by typing the following command line: (the starting speed is specified in quotes).

Attack_Modbus.py "150"

It can be seen that the speed develops as expected and that the engine stops after 30 seconds.



Note: For the attack to work we need to change the firewall configuration. The blocking action of the alarm we had before on the stormshield firewall should be modify to “pass mode”

2. Setting up IP address filtering

The attack is carried out from a PC whose IP is supposed to be distinct from that of the Control PC. In the deployment of this first type of countermeasure, we want to establish a rule authorizing as flows arriving on the ePAC (PLC) only the flows coming from the IP address of this Control PC, and this for all write operations on the ePAC, and excluding all other source IP addresses.

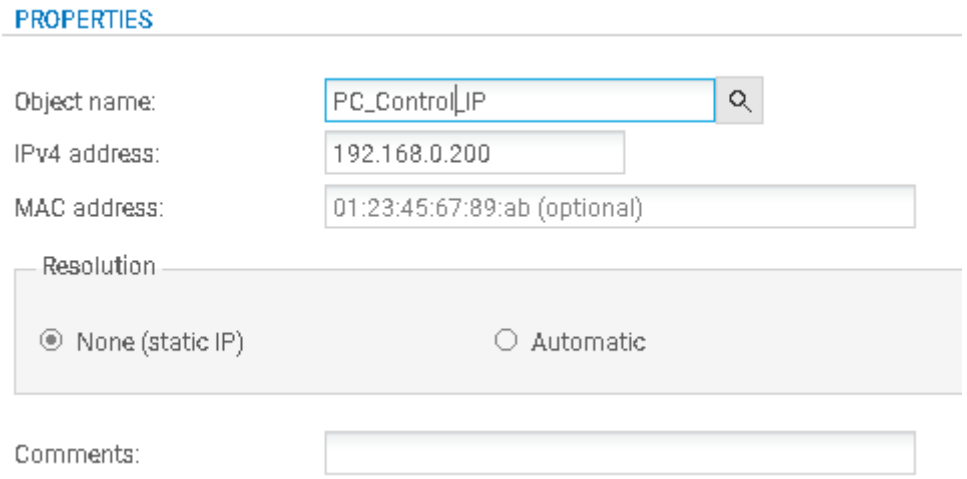
We will therefore consider setting up a specific IP filter (that of the Control PC), in order to prevent the Windows client running the Python script from communicating with the PLC, and therefore from varying the Speed setpoint at the same time as stopping the engine.

The rest of this correction will consider the PC with IP address **192.168.0.200** as the driving PC. (the motorized drive can be concurrently controlled via the HMI)

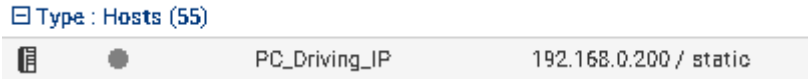
- **Creating a Machine Object**

From **CONFIGURATION | OBJECTS | Network Objects**, we will first create a **Machine** object that we will name **PC_Driving_IP**, and that we will simply define as equivalent to the IP address of the PC legitimately authorized to control the variable speed drive (in the case of this unit test: **192.168.0.200**)

To do this, press the **Add** button to open the following window, where the object (machine) **PC_Driving_IP**, with IPv4 address **192.168.0.200** is created

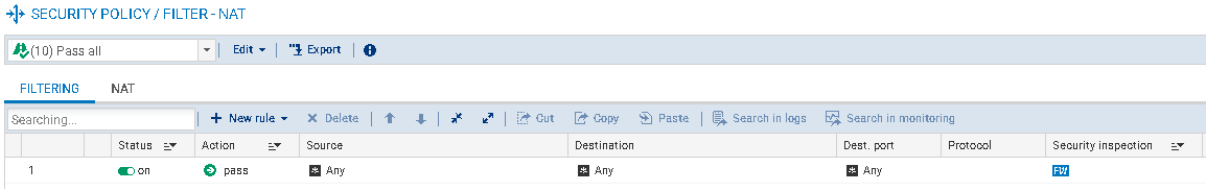


After clicking the **Create** button, our new machine object is displayed in the object list

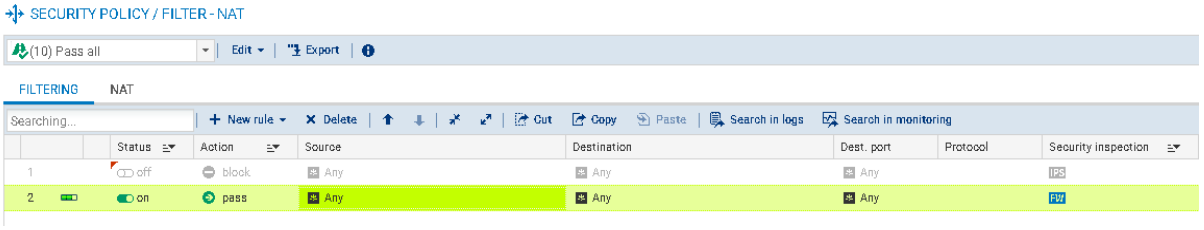


- **Addition of a 'passing' rule**

From **CONFIGURATION | SECURITY POLICY | Filtering and NAT** (see TP #2), we start with a rule that validates the passage and trace of all traffic.

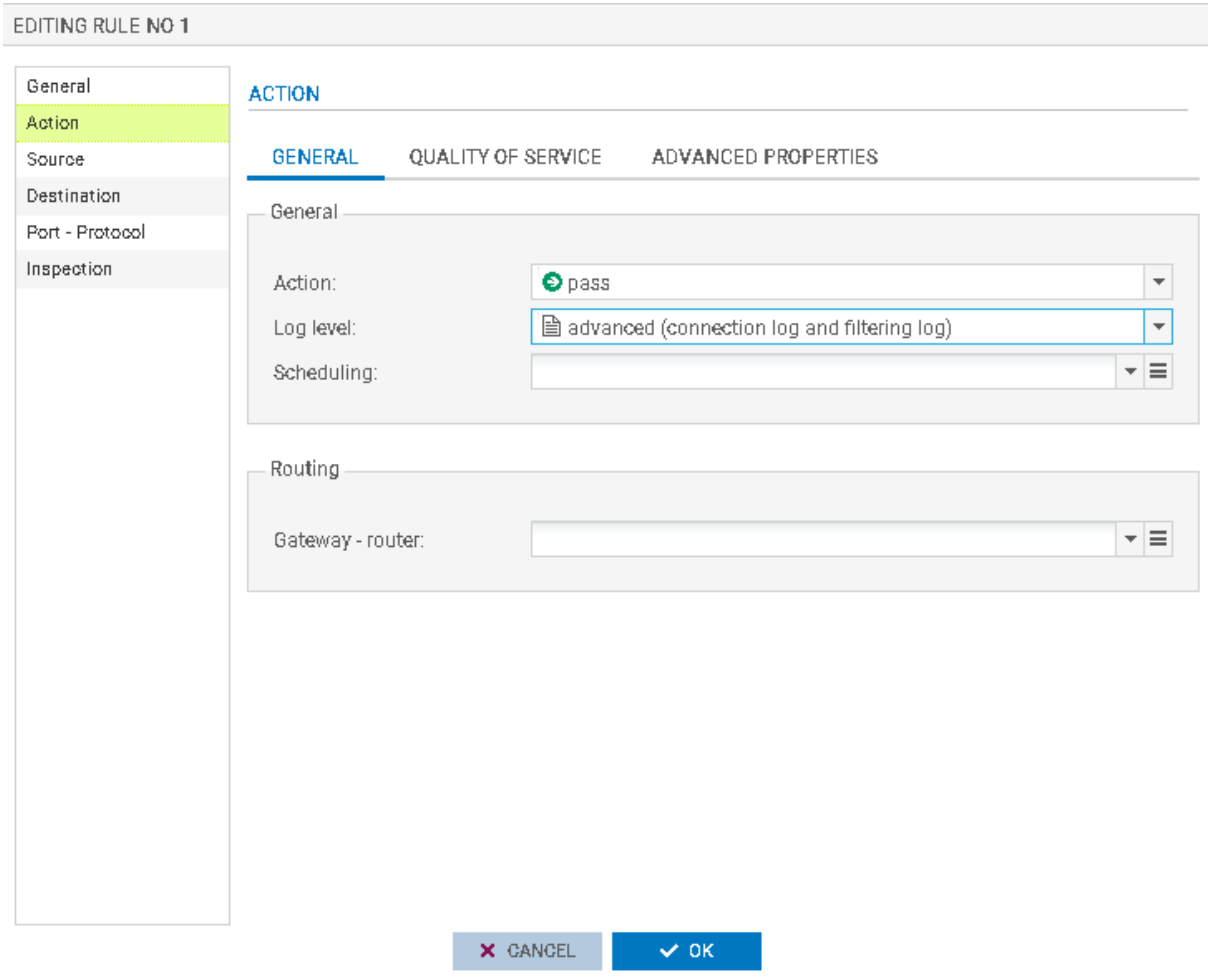


We will add (**New rule**) a simple rule, making sure - by drag'n drop - that this new rule is considered first (otherwise it would not be effective)

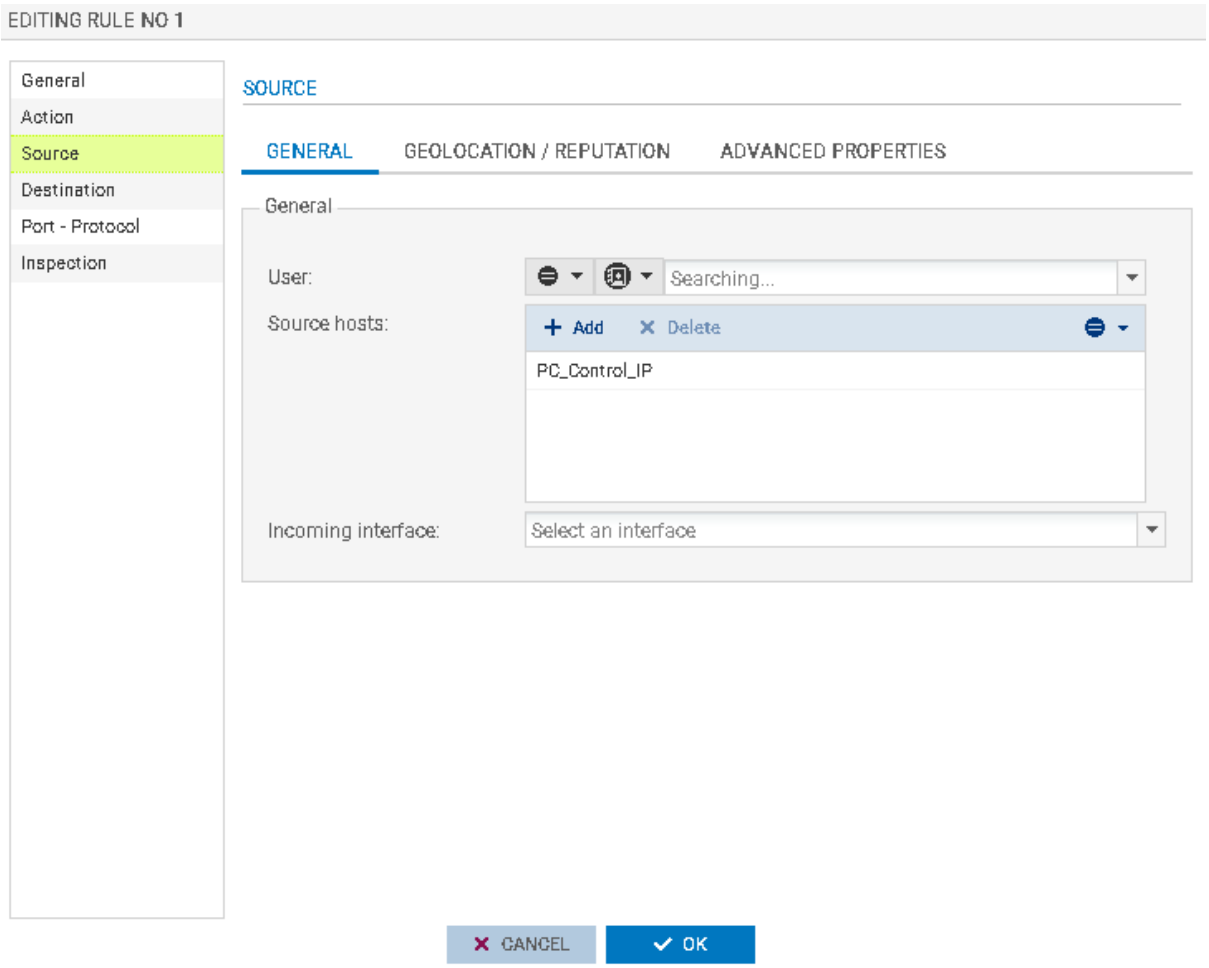


A double click on this rule allows us to specify its parameters:

- In addition to the fact that we will ask for the passage of this flow, we can ask for a detailed trace at the action level, ...



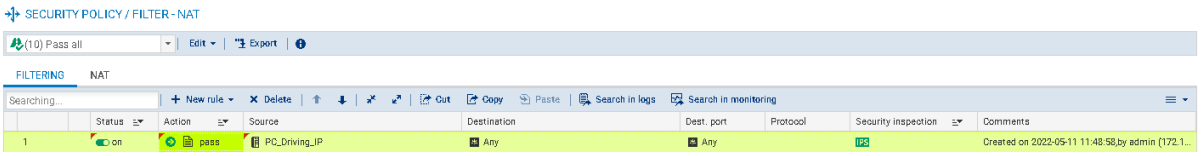
... and designate the PC_Conduite_IP object as the source machine



❖ **Deleting the initial rule**

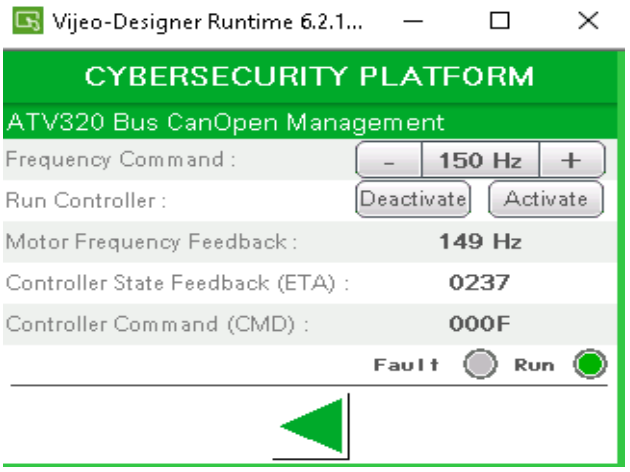
After recording the ON status of this new rule, delete the second rule - the pre-existing one - in order to have a block on anything that does not explicitly come from the Control PC.

After validation of this security policy (which can be renamed if necessary), we obtain

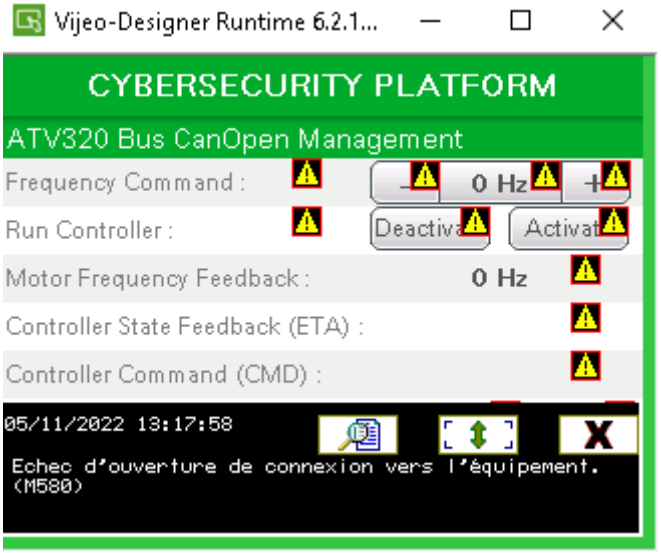


❖ **Test**

We first check that the HMI commands are operational (i.e. commands that do not pass through the firewall), as well as the Modbus commands passing through the firewall (e.g. Vijeo Designer in Simulation mode), issued by the Control PC



By using another PC (or even by changing the IP address of the Workstation, if necessary), the flow is blocked, which of course also applies to an attack initiated using the Python script.




```
c:\Python27>Attack_Modbus.py "150"

Connexion sur port : ('192.168.0.1', 502)
Traceback (most recent call last):
  File "C:\Python27\Attack_Modbus.py", line 23, in <module>
    sock.connect(server_address)
  File "C:\Python27\lib\socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 10060] A connection attempt failed because the connected party did
 not properly respond after a period of time, or established connection failed because
connected host has failed to respond

c:\Python27>_
```

3. Setting up MAC address filtering

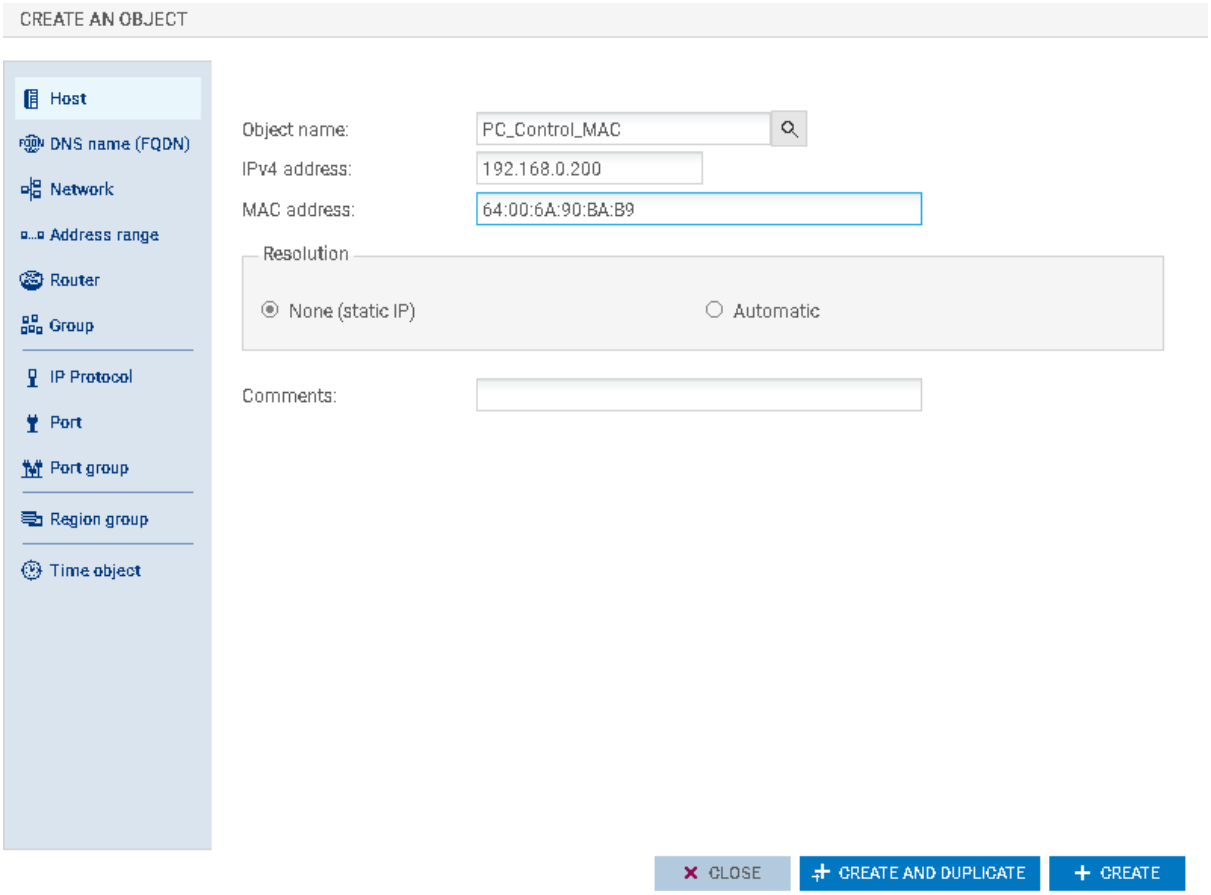
The attack is carried out under identical conditions.

Under the conditions of this unit test, the MAC address of the PC_Control machine used is: (IPCONFIG)

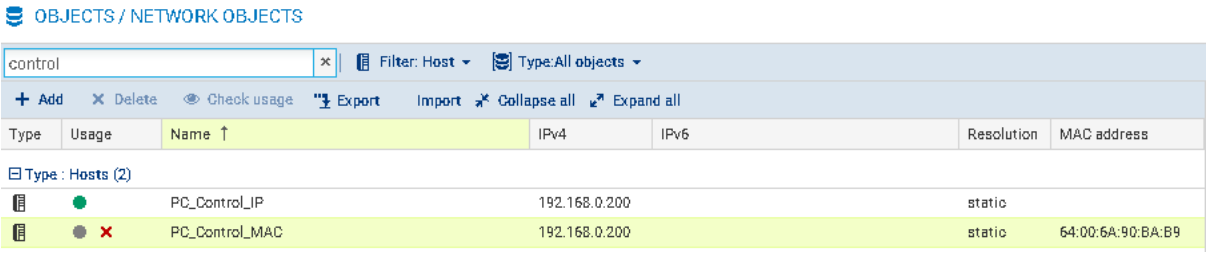
```
Ethernet adapter Connexion au réseau local:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Ethernet Connection (2) I219-LM
   Physical Address. . . . . . . . . : 64-00-6A-90-BA-B9
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::9975:c523:cd71:abc%12(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.0.200(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.254
   DNS Servers . . . . . . . . . . . : 1.1.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

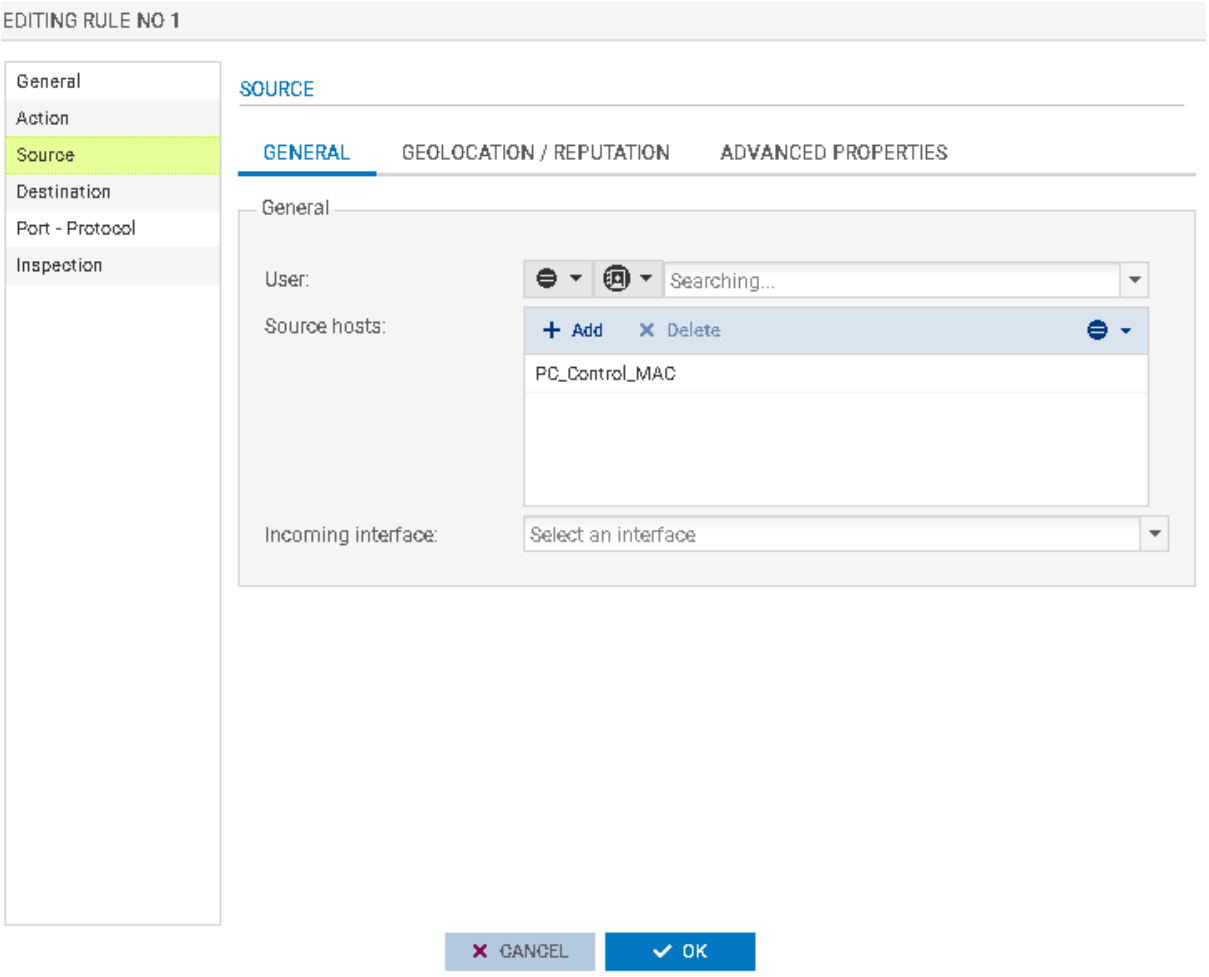
- We return to the definition of the PC_Conduit object, and we will introduce the MAC address of the PC (in this example **64:00:6A:90:BA:B9**), in addition to the IP address, for the definition of a new 'machine' object: **PC_Control_MAC**



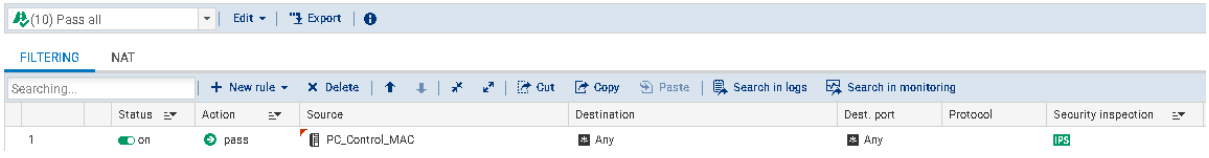
So we now have two PC_Control objects: one **PC_Control_IP** - identified only by its IP address, and the other - **PC_Control_MAC** - also identified by its MAC address



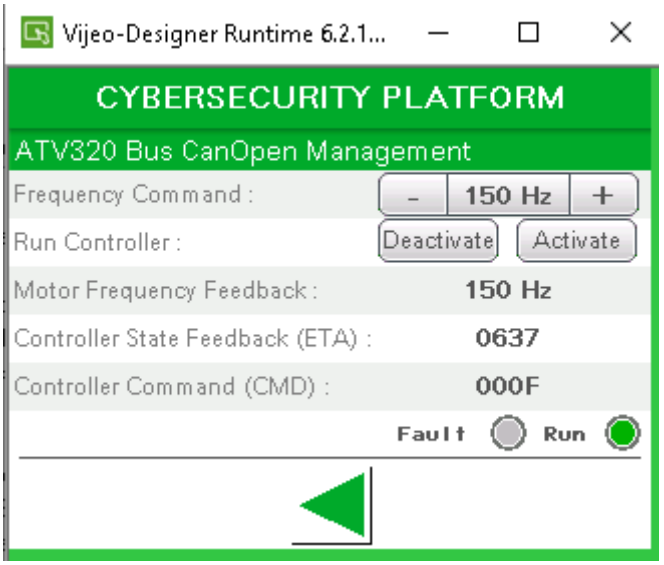
- By now adapting the rule



- The modified rule and associated policy are collected and validated



- Test
  - We first check that the HMI commands are operational (i.e. commands that do not pass through the firewall), as well as the Modbus commands passing through the firewall (e.g. Vijeo Designer in Simulation mode), issued by the Control PC



  - By using another PC (or even by changing the IP address of the Workstation, if necessary), the flow is blocked, which of course also applies to an attack initiated using the Python script.

```
c:\Python27>Attack_Modbus.py "150"

Connexion sur port : ('192.168.0.1', 502)
Traceback (most recent call last):
  File "C:\Python27\Attack_Modbus.py", line 23, in <module>
    sock.connect(server_address)
  File "C:\Python27\lib\socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 10065] A socket operation was attempted to an unreachable host
```