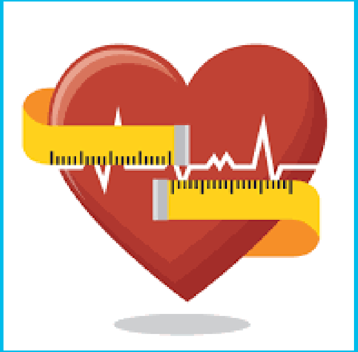


Measuring the health of security devices with Python

Sergey Sazhin – Security Technical Leader in CX. CCIE #37191
Julia Sevostianova - Security Consulting Engineer in CX. CCIE # 53290

05.11.2020

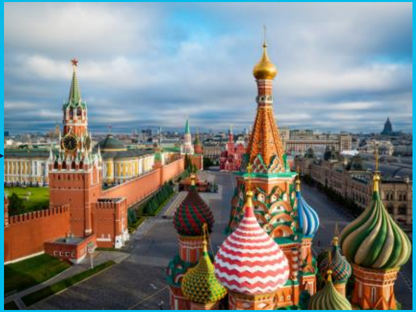
Sergey Sazhin

Work at CX

>7 years at Cisco

At Network Security for 12 years

Based in Moscow



Love snowboarding

Julia Sevostianova

Work at CX

4 years at Cisco

Based in Krakow

Agenda

- ✓ What is PyATS?
- ✓ PyATS overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

Agenda

- ✓ What is PyATS?
- ✓ PyATS overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

Answer in the poll:

What is your programming experience?

What is PyATS?

PyATS – Python library with a set of utilities for automated network testing

Interesting facts:

- Written in Python
- Released internally at Cisco in 2012
- Available on DevNet since 2018
- Used by more than 3500 **developers** and testers inside and outside of Cisco
- Used by Cisco IT
- Works on Linux/macOS*

*On Windows supported on WSL

(Windows Subsystem for Linux)

Where to use PyATS?

NRFU* Tests

CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR

Continuous Deployment
CI/CD**

Troubleshooting

Before/Post upgrade tests

Periodical tests

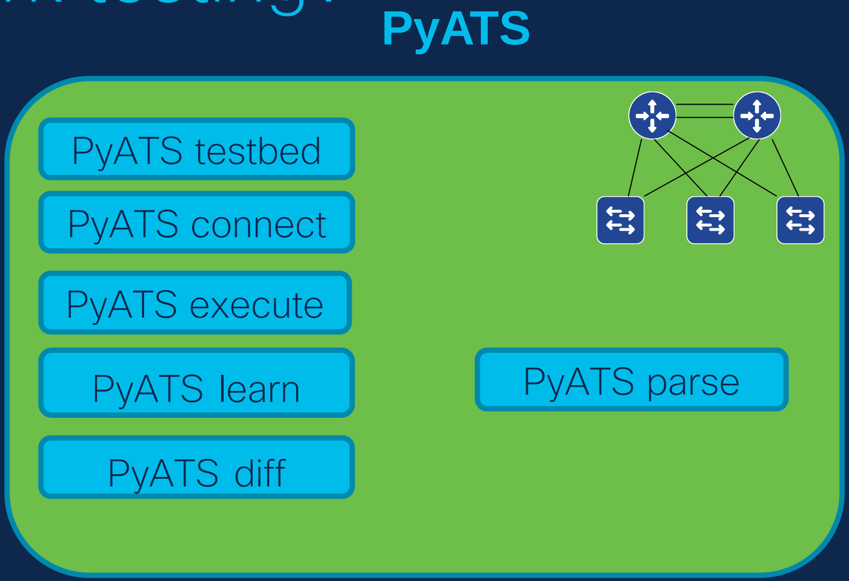
** Network ready for use*
*** CI/CD – Continuous integration and continuous delivery*

Agenda

- ✓ What is PyATS?
- ✓ PyATS overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

What is needed for network testing?

1. Network device list
2. Connect to device
3. Execute test commands
4. Structure output format
5. Compare with expected results
6. Provide report



Topology file for PyATS

Contains information about:

- Devices
- Login credentials
- Connection links

Uses YAML format.

YAML – YAML Ain't Markup Language

- Human readable (hello XML/JSON)
- Easily converted into Python list/dict
- Blocks are formatted with tabs

String:

```
foo: this is a normal string
```

List:

```
# A list of numbers using hyphens:
numbers:
    - one
    - two
    - three
```

Dict:

```
jedi:
  name: Obi-Wan Kenobi
  home-planet: Stewjon
  species: human
  master: Qui-Gon Jinn
  height: 1.82m
```

Example of topology file for PyATS

```
$ cat testbed.yaml
```

```
name: labpyats

 credentials:
   default:
     username: "%ENV{PYATS_USERNAME}"
     password: "%ENV{PYATS_PASSWORD}"
   enable:
     password: "%ENV{PYATS_AUTH_PASS}"
   line:
     password: "%ENV{PYATS_AUTH_PASS}"
```

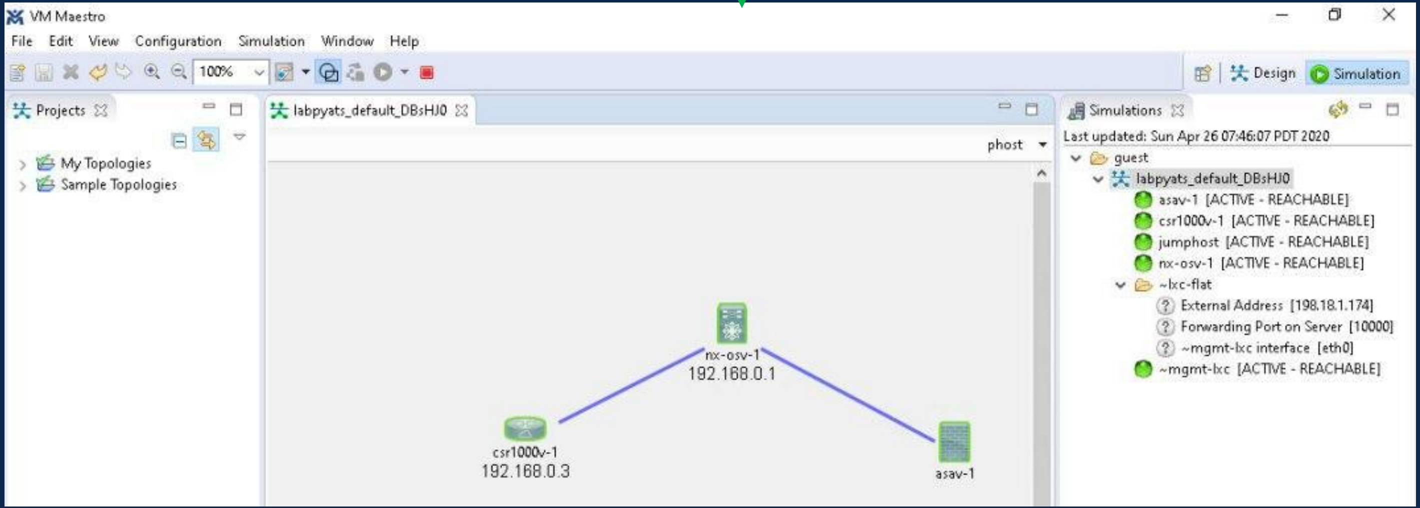
```
  EdgeFW:
      alias: edgefw
      os: asa
      type: ASAv
      platform: ASAv

      connections:

        console:
          protocol: ssh
          ip: 172.16.50.191
          port: 22
```

name – topology name
credentials – login details (stored in environment variables)

EdgeFW – name of device
os/type/platform – device type (for connection establishment)
connections -how to connect

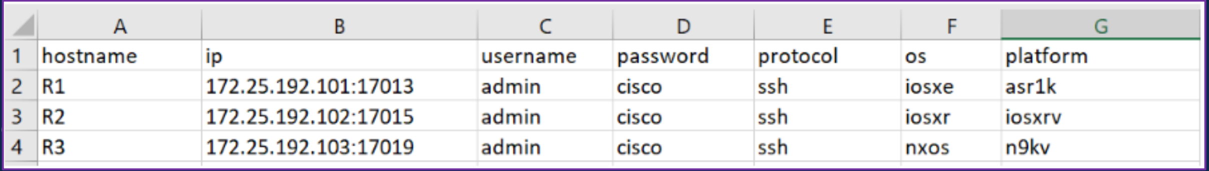
How to get topology file?

- VIRL-file (virlutils)
- EXCEL-table



Last updated: Sun Apr 26 07:46:07 PDT 2020
- guest
  - labpyats_default_DBsHJ0
    - asav-1 [ACTIVE - REACHABLE]
    - csr1000v-1 [ACTIVE - REACHABLE]
    - jumphost [ACTIVE - REACHABLE]
    - nx-osv-1 [ACTIVE - REACHABLE]
    - ~lxc-flat
      - External Address [198.18.1.174]
      - Forwarding Port on Server [10000]
      - ~mgmt-lxc interface [eth0]
    - ~mgmt-lxc [ACTIVE - REACHABLE]

```
pyats create testbed my_devices.xls --output testbed.yaml
```



	A	B	C	D	E	F	G
1	hostname	ip	username	password	protocol	os	platform
2	R1	172.25.192.101:17013	admin	cisco	ssh	iosxe	asr1k
3	R2	172.25.192.102:17015	admin	cisco	ssh	iosxr	iosxrv
4	R3	172.25.192.103:17019	admin	cisco	ssh	nxos	n9kv

Answer in the poll:

What format is used in PyATS topology file?

What about 3rd party?

Supported Cisco platforms:

- IOS
- NX-OS
- IOS XR
- ASA
- FTD
- 3rd party*

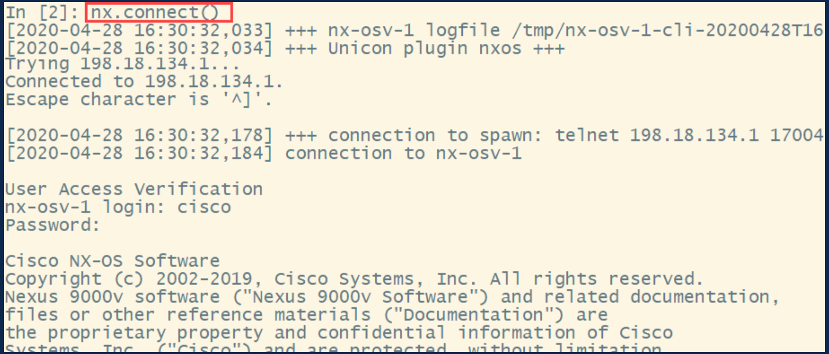
https://pubhub.devnetcloud.com/media/unicon/docs/user_guide/supported_platforms.html

Plug-in list:

https://github.com/CiscoTestAutomation/unicon.plugins

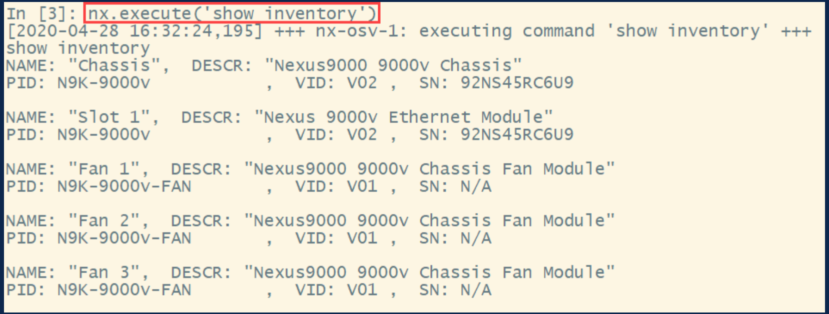
Connecting to devices

Pyats connect – establish connection



```
In [2]: nx.connect()
[2020-04-28 16:30:32,033] +++ nx-osv-1 logfile /tmp/nx-osv-1-cli-20200428T16
[2020-04-28 16:30:32,034] +++ Unicon plugin nxos +++
Trying 198.18.134.1...
Connected to 198.18.134.1.
Escape character is '^]'.

[2020-04-28 16:30:32,178] +++ connection to spawn: telnet 198.18.134.1 17004
[2020-04-28 16:30:32,184] connection to nx-osv-1

User Access Verification
nx-osv-1 login: cisco
Password:

Cisco NX-OS Software
Copyright (c) 2002-2019, Cisco Systems, Inc. All rights reserved.
Nexus 9000v software ("Nexus 9000v Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation
```

Pyats execute – pass a command and get an output in string format



```
In [3]: nx.execute('show inventory')
[2020-04-28 16:32:24,195] +++ nx-osv-1: executing command 'show inventory' +++
show inventory
NAME: "Chassis",  DESCR: "Nexus9000 9000v Chassis"
PID: N9K-9000v            ,  VID: V02 ,  SN: 92NS45RC6U9

NAME: "Slot 1",  DESCR: "Nexus 9000v Ethernet Module"
PID: N9K-9000v            ,  VID: V02 ,  SN: 92NS45RC6U9

NAME: "Fan 1",  DESCR: "Nexus9000 9000v Chassis Fan Module"
PID: N9K-9000v-FAN        ,  VID: V01 ,  SN: N/A

NAME: "Fan 2",  DESCR: "Nexus9000 9000v Chassis Fan Module"
PID: N9K-9000v-FAN        ,  VID: V01 ,  SN: N/A

NAME: "Fan 3",  DESCR: "Nexus9000 9000v Chassis Fan Module"
PID: N9K-9000v-FAN        ,  VID: V01 ,  SN: N/A
```

PyATS CLI

- Some PyATS utilities can be run from Linux shell
- For example: pyats parse, learn, diff
- No coding experience required

Answer in the poll: Choose an example in JSON format

```
{ "users" : [ {
"name" : "Alice" ,
age : 20 } ] }
```

#1

```
<users>
  <user name="Alice">
    <age>20</age>
  </user>
</users>
```

#2

```
users:
  Alice:
    age: 18
```

#3

```
Username;Age
Alice;18
```

#4

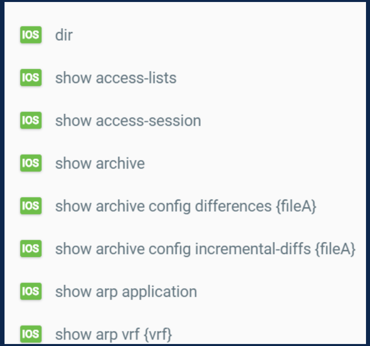
PyATS parse

Parsing – collecting non-structured data (text) and saving it JSON format

$$\begin{tikzcd}\boxed{\text{Show-command}} \arrow[r, "\text{PyATS parse}"] & \boxed{\text{Python dictionary}}\end{tikzcd}$$

- Pre-configured parsers for:
- IOS
- IOS XR
- NX-OS
- ASA
- ...

1600 parsers in total!



IOS dir

IOS show access-lists

IOS show access-session

IOS show archive

IOS show archive config differences {fileA}

IOS show archive config incremental-diffs {fileA}

IOS show arp application

IOS show arp vrf {vrf}

https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/parsers

Answer for the poll: Option #1 is in JSON format

```
{ "users" : [ {
"name" : "Alice" ,
age : 20 } ] }
```

#1

```
<users>
  <user name="Alice">
    <age>20</age>
  </user>
</users>
```

#2

```
users:
  Alice:
    age: 18
```

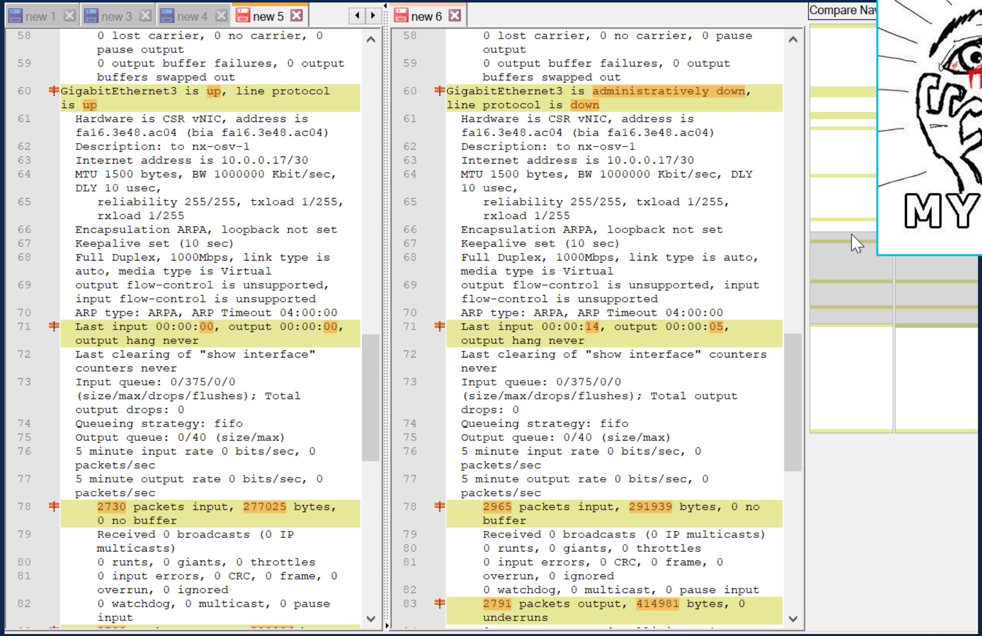
#3

```
Username;Age
Alice;18
```

#4

Comparing text files

Example (common approach)



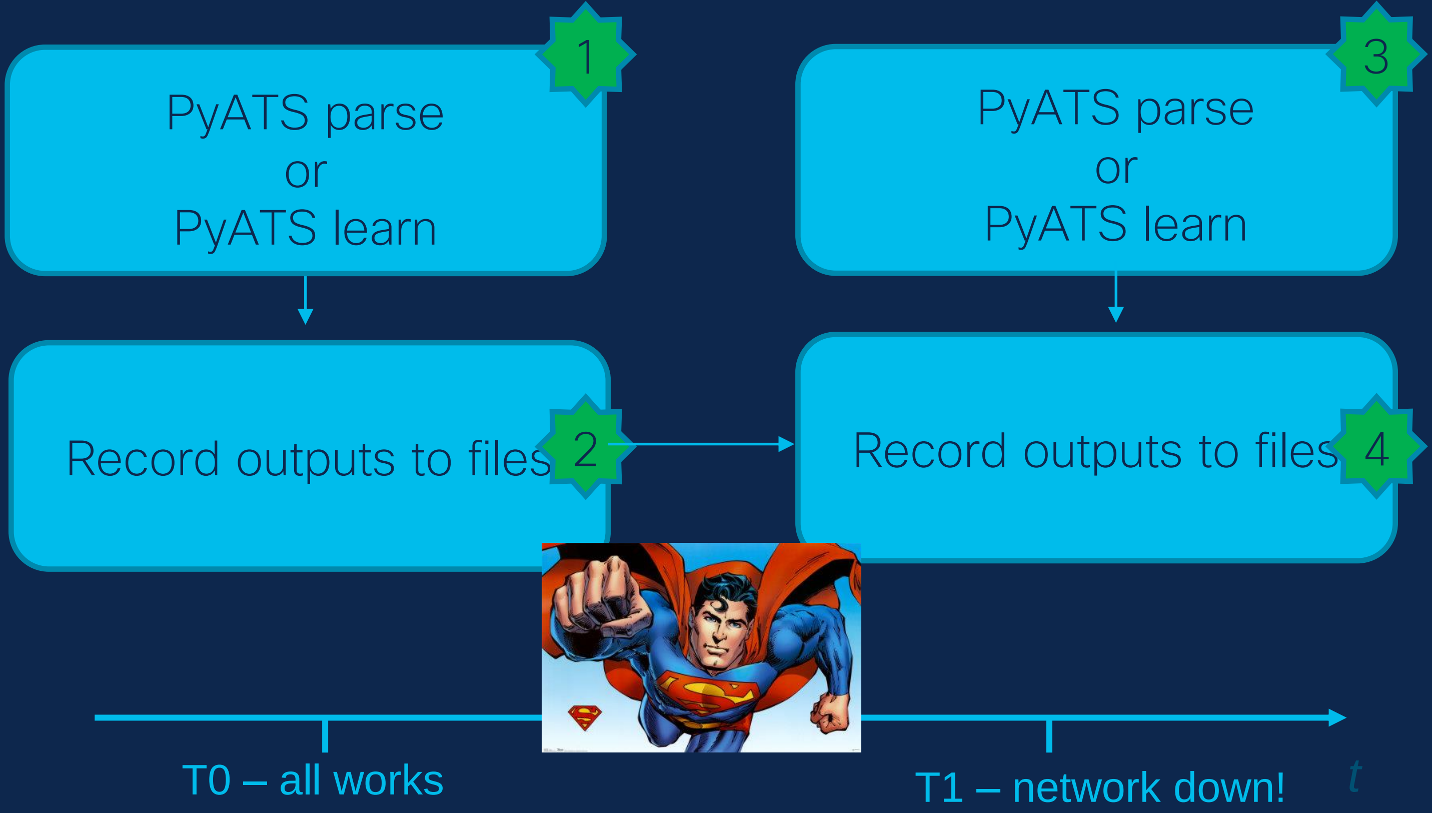
```
        0 lost carrier, 0 no carrier, 0 pause output
        0 output buffer failures, 0 output buffers swapped out
GigabitEthernet3 is up, line protocol is up
     Hardware is CSR vNIC, address is fa16.3e48.ac04 (bia fa16.3e48.ac04)
     Description: to nx-osv-1
     Internet address is 10.0.0.17/30
     MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
        reliability 255/255, txload 1/255, rxload 1/255
     Encapsulation ARPA, loopback not set
     Keepalive set (10 sec)
     Full Duplex, 1000Mbps, link type is auto, media type is Virtual
     output flow-control is unsupported, input flow-control is unsupported
     ARP type: ARPA, ARP Timeout 04:00:00
     Last input 00:00:00, output 00:00:00, output hang never
     Last clearing of "show interface" counters never
     Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
     Queueing strategy: fifo
     Output queue: 0/40 (size/max)
     5 minute input rate 0 bits/sec, 0 packets/sec
     5 minute output rate 0 bits/sec, 0 packets/sec
        2730 packets input, 277025 bytes, 0 no buffer
        Received 0 broadcasts (0 IP multicasts)
        0 runts, 0 giants, 0 throttles
        0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
        0 watchdog, 0 multicast, 0 pause input
```

```
        0 lost carrier, 0 no carrier, 0 pause output
        0 output buffer failures, 0 output buffers swapped out
GigabitEthernet3 is administratively down, line protocol is down
     Hardware is CSR vNIC, address is fa16.3e48.ac04 (bia fa16.3e48.ac04)
     Description: to nx-osv-1
     Internet address is 10.0.0.17/30
     MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
        reliability 255/255, txload 1/255, rxload 1/255
     Encapsulation ARPA, loopback not set
     Keepalive set (10 sec)
     Full Duplex, 1000Mbps, link type is auto, media type is Virtual
     output flow-control is unsupported, input flow-control is unsupported
     ARP type: ARPA, ARP Timeout 04:00:00
     Last input 00:00:14, output 00:00:05, output hang never
     Last clearing of "show interface" counters never
     Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
     Queueing strategy: fifo
     Output queue: 0/40 (size/max)
     5 minute input rate 0 bits/sec, 0 packets/sec
     5 minute output rate 0 bits/sec, 0 packets/sec
        2965 packets input, 291939 bytes, 0 no buffer
        Received 0 broadcasts (0 IP multicasts)
        0 runts, 0 giants, 0 throttles
        0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
        0 watchdog, 0 multicast, 0 pause input
        2791 packets output, 414981 bytes, 0 underruns
```

T0 – everything works

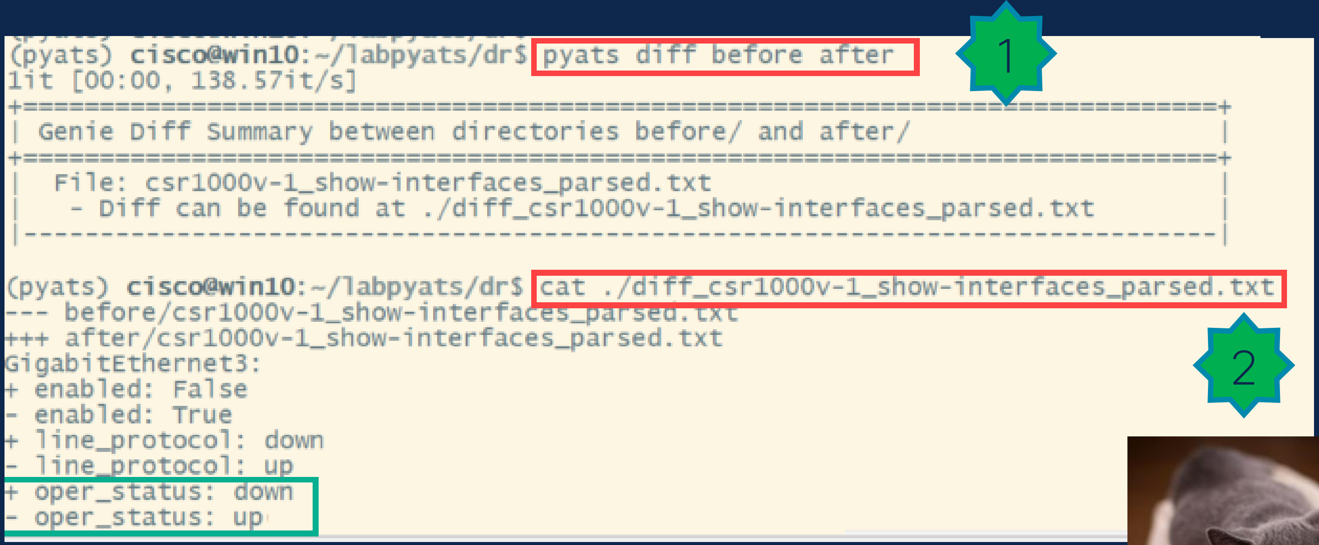
T1 – network down!

PyATS diff


1
PyATS parse
or
PyATS learn
2
Record outputs to files
3
PyATS parse
or
PyATS learn
4
Record outputs to files
T0 – all works
T1 – network down!
t

PyATS diff

- Not just comparing files before and after
- Compare parsing results in folders ‘before’ & ‘after’
- Presents only relevant difference:



```
(pyats) cisco@win10:~/labpyats/dr$ pyats diff before after
1it [00:00, 138.57it/s]
+==============================================================================+
| Genie Diff Summary between directories before/ and after/                    |
+==============================================================================+
|  File: csr1000v-1_show-interfaces_parsed.txt                                 |
|   - Diff can be found at ./diff_csr1000v-1_show-interfaces_parsed.txt       |
|------------------------------------------------------------------------------|

(pyats) cisco@win10:~/labpyats/dr$ cat ./diff_csr1000v-1_show-interfaces_parsed.txt
--- before/csr1000v-1_show-interfaces_parsed.txt
+++ after/csr1000v-1_show-interfaces_parsed.txt
GigabitEthernet3:
+ enabled: False
- enabled: True
+ line_protocol: down
- line_protocol: up
+ oper_status: down
- oper_status: up
```

Agenda

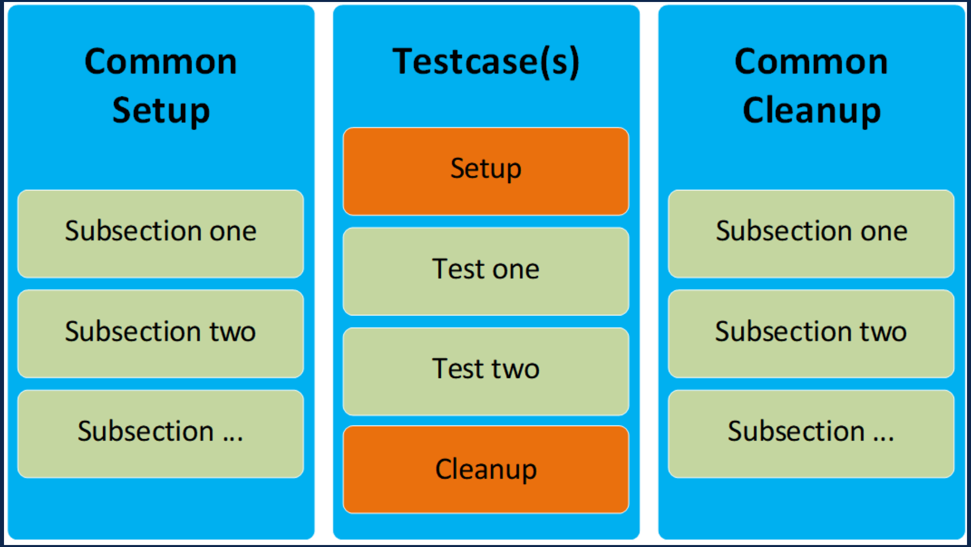
- ✓ What is PyATS?
- ✓ PyATS tools overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

Answer in the poll:

What is your experience with object oriented programming (OOP)?

PyATS test structure

- Tests are written in Python
- Each section is a separate class
- Understanding of Object Oriented Programming is recommended



RUN TEST!!!!!

RRUUUNN!!!!!

Order of execution in Python script

PyATS test structure: Python code

Common Setup

```
class MyCommonSetup(aetest.CommonSetup):
<before testing: loading testbed connecting to devices.
Outputs passed to all tests>
```

Testcases

```
class Testcase1(aetest.Testcase):
<tests logic>
..
class TestcaseN(aetest.Testcase):
<tests logic>
```

Common Cleanup

```
class MyCommonCleanup(aetest.CommonCleanup):
<after testing: revert all changes in Common Setup or
tests>
```

Start with creating a Test (Common Setup)

empty_pyats_example.py

```
class common_setup(aetest.CommonSetup):

    @aetest.subsection
    def establish_connections(self, testbed):
        # Load testbed file which is passed as command-line argument
        genie_testbed = Genie.init(testbed)
        # Load devices from testbed and try to connect to one of them
        device = genie_testbed.devices['vpnfw']
        log.info(banner(f"Connect to device '{device.name}'"))
        try:
            device.connect(log_stdout=False)
        except errors.ConnectionError:
            self.failed("Failed to establish connection to device")
        self.parent.parameters['device'] = device


class sample_test(aetest.Testcase):
    @aetest.test
    def test_parse(self):
        device = self.parent.parameters['device']
        log.info(device.parse('show asp drop'))
```

section

subsection

section

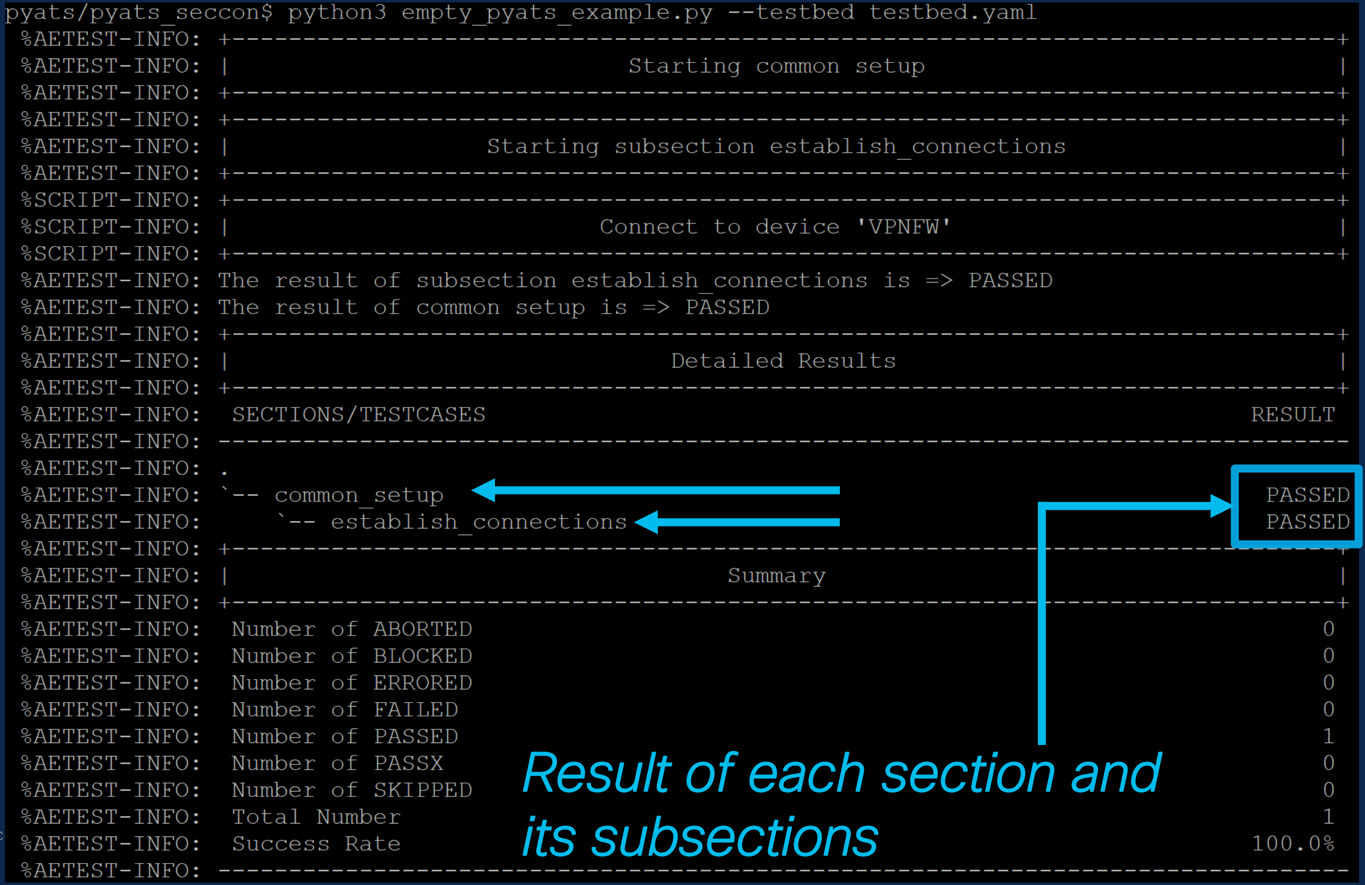
test

Run the test:

```
# python3 empty_pyats_example.py --testbed testbed.yaml
```

Test results (CLI)

empty_pyats_example.py



```
pyats/pyats_seccon$ python3 empty_pyats_example.py --testbed testbed.yaml
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: |                            Starting common setup                             |
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: |                   Starting subsection establish_connections                  |
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %SCRIPT-INFO: +------------------------------------------------------------------------------+
 %SCRIPT-INFO: |                          Connect to device 'VPNFW'                           |
 %SCRIPT-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: The result of subsection establish_connections is => PASSED
 %AETEST-INFO: The result of common setup is => PASSED
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: |                               Detailed Results                               |
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO:  SECTIONS/TESTCASES                                                      RESULT
 %AETEST-INFO: -------------------------------------------------------------------------------
 %AETEST-INFO: .
 %AETEST-INFO: `-- common_setup                                                         PASSED
 %AETEST-INFO:     `-- establish_connections                                            PASSED
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO: |                                   Summary                                    |
 %AETEST-INFO: +------------------------------------------------------------------------------+
 %AETEST-INFO:  Number of ABORTED                                                            0
 %AETEST-INFO:  Number of BLOCKED                                                            0
 %AETEST-INFO:  Number of ERRORED                                                            0
 %AETEST-INFO:  Number of FAILED                                                             0
 %AETEST-INFO:  Number of PASSED                                                             1
 %AETEST-INFO:  Number of PASSX                                                              0
 %AETEST-INFO:  Number of SKIPPED                                                            0
 %AETEST-INFO:  Total Number                                                                 1
 %AETEST-INFO:  Success Rate                                                             100.0%
 %AETEST-INFO: -------------------------------------------------------------------------------
```

Result of each section and its subsections

Agenda

- ✓ What is PyATS?
- ✓ PyATS overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

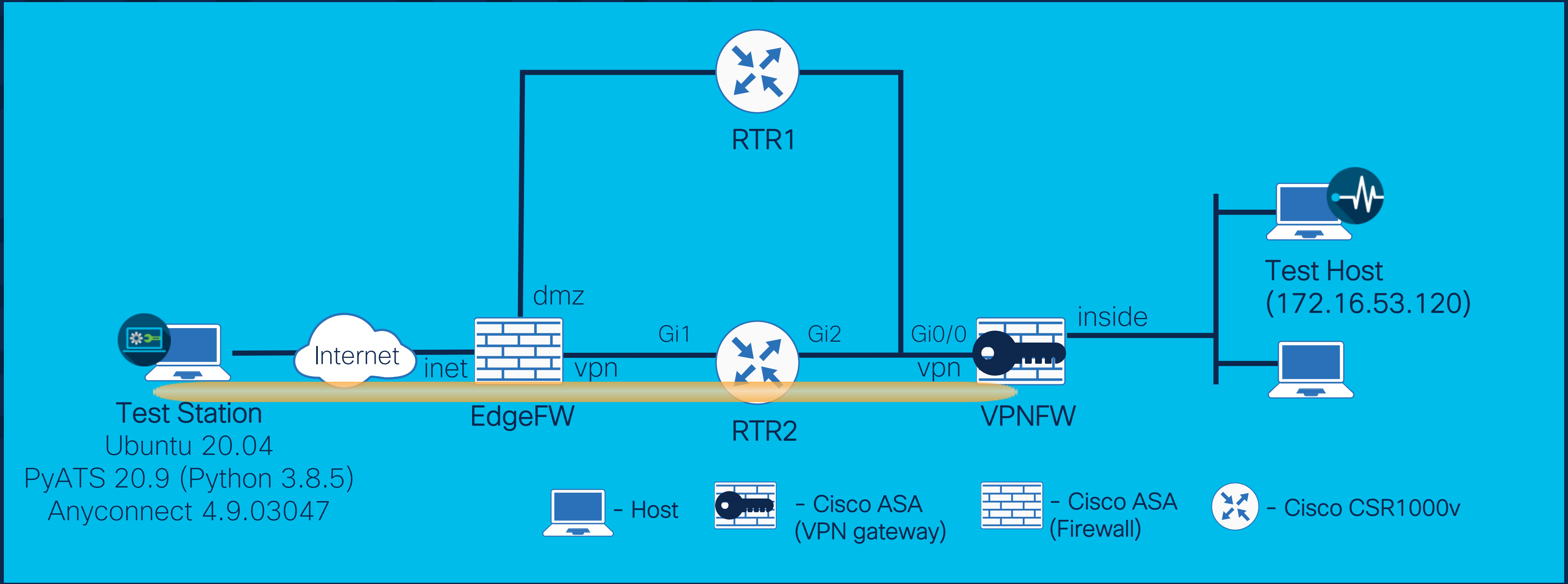
Demo Time
Automation of health checks for security devices

Let's see the power of pyATS in action

Meow! Show me how!

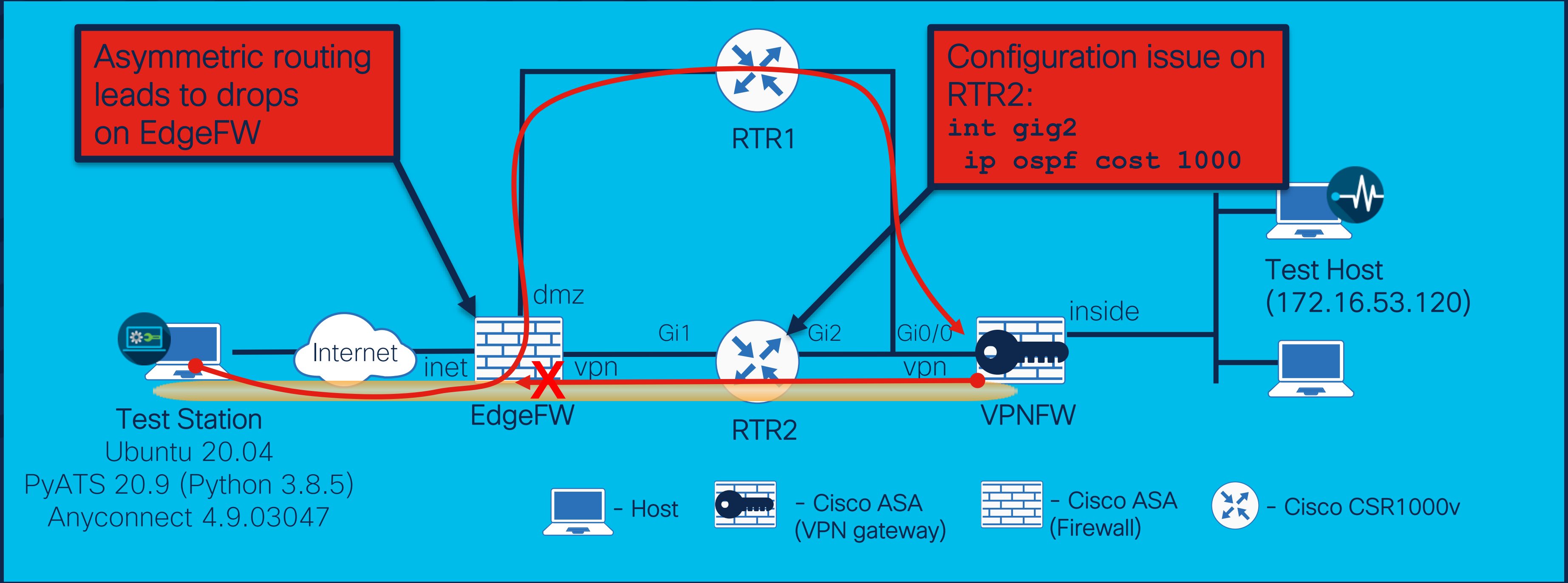
Demo Time

Network Topology



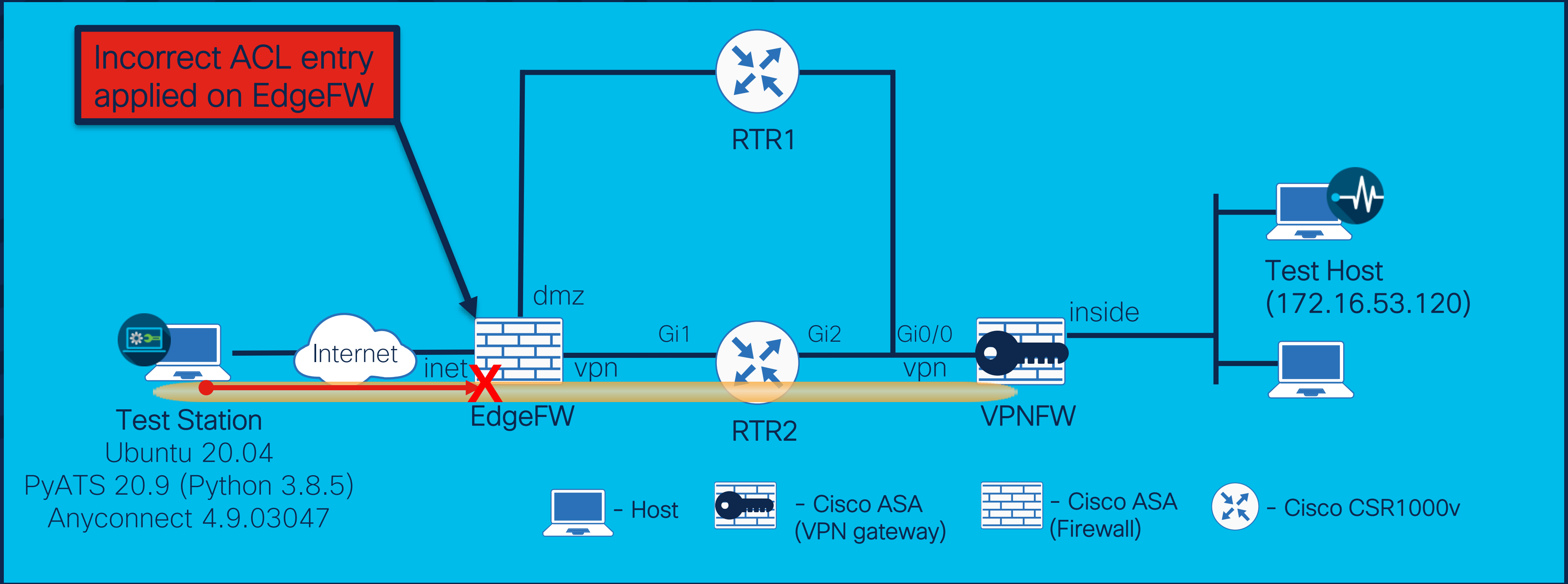
Demo Time

Demo Part #5 – Routing issue



Demo Time

Demo Part #6 – Issue with ACL on EdgeFW



Agenda

- ✓ What is PyATS?
- ✓ PyATS overview
- ✓ Testing with PyATS
- ✓ DEMO
- ✓ Conclusion and the next steps

Conclusion

- pyATS could be used for getting quick posture of the network
- Sky is the limit since you can use Python to extend this library
- You can use pyATS to measure health of your security devices

You can test pyATS right after the session!

Support alias:
pyats-support@cisco.com

Checkout the code from demo on GitHub:
https://github.com/sesazhin/pyats_seccon.git

Webex Teams (internal) room:
https://eurl.io/#AmB-3s-d

Quick start for PyATS

DEVNET lab:
https://developer.cisco.com/learning/lab/intro-to-pyats/step/1

Download Docker image with PyATS:
https://github.com/CiscoTestAutomation/pyats-docker

Or install pyATS on Linux/macOS:
https://developer.cisco.com/docs/pyats-getting-started/

Useful information for quick start:
https://github.com/CiscoTestAutomation/getting-started/tree/master/start-guide

Pass pyATS lab on dCloud:
https://dcloud2-lon.cisco.com/content/demo/428412