Aprendizagem Aplicada à Segurança

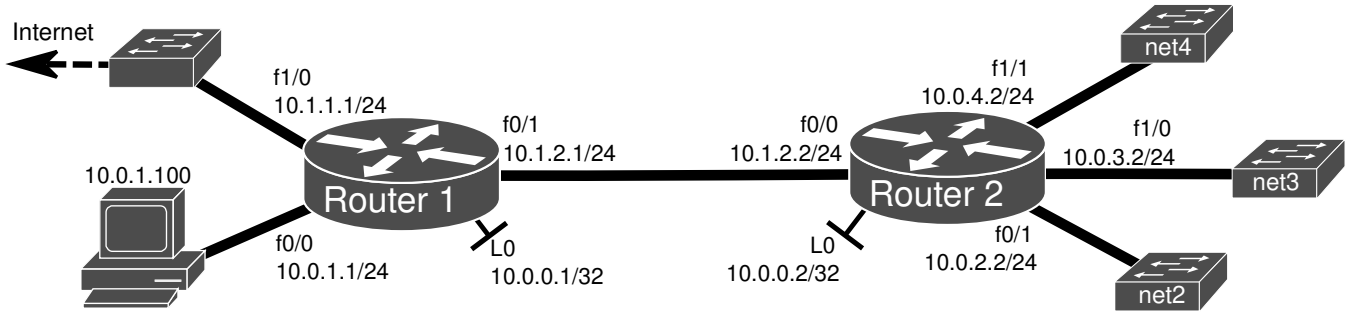
Data Acquisition

Objectives

- Nodes, links and terminals data acquisition with SNMP
- Traffic flows data acquisition with Netflow/IPFIX
- Equipment status and events acquisition via SSH
- Server/service events acquisition via SCP
- Raw traffic acquisition with libpcap

Data acquisition with SNMP

1. Configure a network (in GNS3) according to the following figure. The PC can be a VM or the host PC, with Linux (Debian) with Python, SNMP tools, network MIBs and CISCO MIBs. The Routers should be from the 7200 family.



MIB references: MIBS: IF-MIB, IP-MIB, and CISCO-QUEUE-MIB

Python references: *Snimpy* (version>=0.8.14) – API reference, https://snimpy.readthedocs.org/en/latest/api.html
argparse - Parser for command-line options, https://docs.python.org/3/library/argparse.html
matplotlib.pyplot - http://matplotlib.org/api/pyplot_api.html

Configuration example for Router 1 (only interfaces f0/0 and loopback 0):

```
Router# configure terminal
Router(config)# interface f0/0
Router(config-if)# ip address 10.0.1.1 255.255.255.0
Router(config-if)# ip ospf 1 area 0
Router(config-if)# no shutdown
Router(config-if)# interface loopback 0
Router(config-if)# ip address 10.0.0.1 255.255.255.255
Router(config-if)# ip ospf 1 area 0
Router(config-if)# no shutdown
Router(config-if)# end
Router(config-if)# write
```

2. In both routers, configure a SNMP version 3 community (using the name “private”) with Read-Only permissions, and access with authentication (MD5, password authpass) and encryption (AES128, password: privpass), for user uDDR from group gDDR:

```
Router(config)# snmp-server user uDDR gDDR v3 auth md5 authpass priv aes 128 privpass
Router(config)# snmp-server group gDDR v3 priv
Router(config)# snmp-server community private RO
```

3. Download and test the baseSNMP.py script, and understand how different MIB objects can be accessed.

```
python baseSNMP.py -r 10.0.0.2
```

4. Use the following MIB objects to access relevant interface traffic statistics:
- *ifHCOutUcastPkts*, *ifHCInUcastPkts*, *ifHCOutOctets*, *ifHCInOctets* from IF-MIB.

5. Create a time loop, with periodicity given by argument, and retrieve interface statistics. Display, and store, byte and packet increments (in both directions) with timestamp.

(Optional) Data acquisition with NetFlow/IPFIX

NetFlow references: NetFlow Overview

NetFlow Export Datagram Format (version 1 and 5 formats)

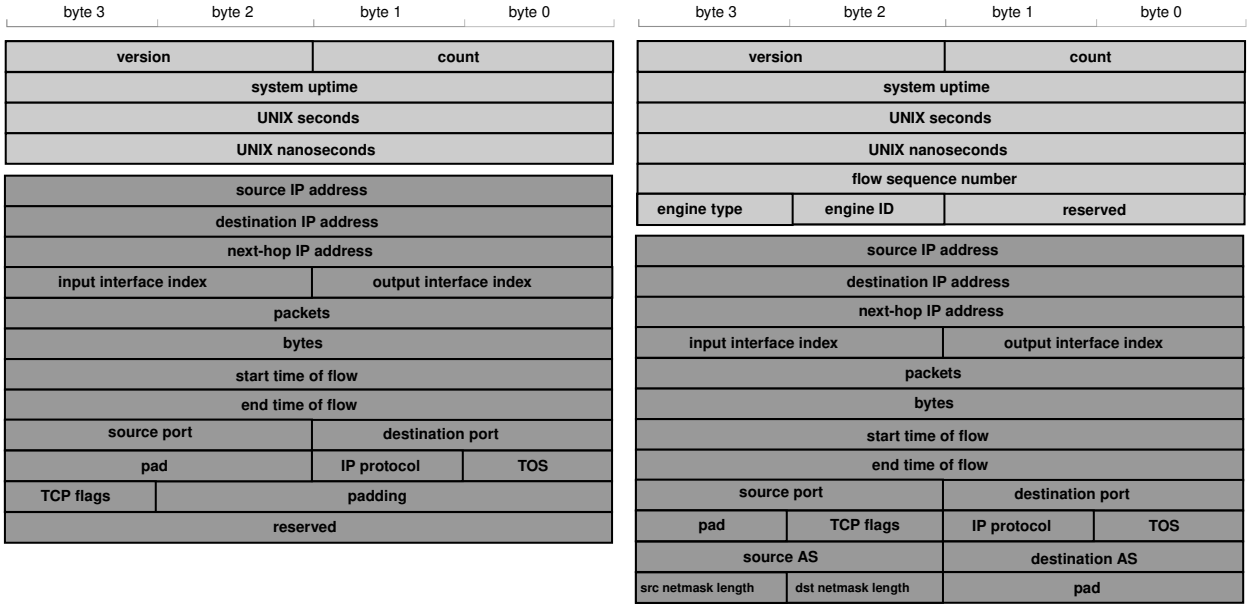
Python references: *socket* - Low-level networking interface, https://docs.python.org/2/library/socket.html

struct - Interpret strings as packed binary data, https://docs.python.org/2/library/struct.html

netaddr IPAddress and IPNetwork - https://netaddr.readthedocs.org/en/latest/tutorial_01.html

netaddr IPSet - https://netaddr.readthedocs.org/en/latest/tutorial_03.html

NetFlow v1 and v5 header and body formats



6. Configure Router2 to export (to PC VM) the flow statistics using NetFlow version 1 for all traffic egressing interface f0/1.

```
Router2(config)# interface FastEthernet0/1
Router2(config-if)# ip flow egress
Router2(config)# ip flow-export destination 10.0.1.100 9996
Router2(config)# ip flow-export source Loopback 0
Router2(config)# ip flow-export version 1
```

7. Download and test the baseNetFlow.py script, generating traffic to and from terminals (VPCS) in networks net2, net3, and net4. Understand how the NetFlow packet is received and how data fields can be accessed.

```
python baseNetFlow.py -r 10.0.0.2 -n 10.0.2.0/24 10.0.3.0/24 10.0.4.0/24
```

8. Complete the code to retrieve the relevant NetFlow version 1 data and periodically infer the traffic matrix. Configure the required additional NetFlow export commands in Router1 and Router2.

Note: consider using Python library *netaddr* classes IPAddress, IPNetwork, and IPSet.

Note2: consider the output format (line):

```
timestamp_1, Trf_Net1_Net2,  Trf_Net1_Net3, …, Trf_Net4_Net2, Trf_Net4_Net3
timestamp_2, Trf_Net1_Net2,  Trf_Net1_Net3, …, Trf_Net4_Net2, Trf_Net4_Net3
...
```

9. Include support to NetFlow version 5.

Change routers' configurations to export flow data using NetFlow version 5:

```
Router(config)# ip flow-export version 5
```

Data acquisition via CLI and SSH

10. To the network constructed in 1., add and configure a SWL3 device in net2 *and activate the Routers' and SWL3 remote conso*le via SSH.

```
Router1(config)# aaa new-model
Router1(config)# enable password labcom
Router1(config)# username labcom secret 0 labcom
Router1(config)# ip domain-name con.ara.com
Router1(config)# ip ssh rsa keypair-name sshkey
Router1(config)# crypto key generate rsa usage-keys label sshkey modulus 2048
Router1(config)# ip ssh version 2
Router1(config)# ip ssh time-out 60
Router1(config)# ip ssh authentication-retries 2
```

Download, test and improve the baseSSHConsole.py by retrieving the ARP tables from all devices, and the Forwarding Table from the SWL3. Construct a basic rule to detect MAC and IP Spoofing attacks.

The relevant Cisco IOS commands are:

```
Router# show arp
Router# show mac-address-table
```

Note: to test the SSH connection from a Linux terminal (the additional ssh parameters are required because Cisco's IOS does not initiate cryptography negotiations):

```
ssh -oKexAlgorithms=+diffie-hellman-group1-sha1 -oHostKeyAlgorithms=+ssh-rsa -c aes256-cbc labcom@10.0.1.1
```

11. To the network constructed in 1., add a Linux server VM (with SSH server). Based on the baseSSHConsole.py script, remotely retrieve the server CPU and memory usage. Relevant bash commands:

```
top -n1 -1
mpstat
cat /proc/meminfo
```

Data acquisition from Service Logs

12. At the Linux server guarantee that services Apache and DNS (bind9/named) are running. Using as base the baseLogServer.py script that fetches the local Apache2 log and parses events, define, list ans store (timestamped) specific events based on content or client group (IPv4 network).

13. Improve the baseLogServer.py script to fetch the local DNS log and parse events. By default bind9/named services output to /var/log/syslog (entries marked with label named). List and store (timestamped) specific requests from a specific client (IPv4 address) or client group (IPv4 network).

14. Change the baseLogServer.py script to perform the service log event parsing remotely, using SCP (it is assumed that the server as an active SSH server).

Data acquisition with pcap

Python references: pyshark - Python packet parser using wireshark's tshark, http://kiminewt.github.io/pyshark/

- https://github.com/KimiNewt/pyshark

15. Download and test the basePCap.py script, by capturing all IPv4 packets between a source and destination passed by argument. Source and destination should be an network prefix (no mask for a single machine):

```
python basePCap.py -i eth0 -c <pc_ipaddr> -s 0.0.0.0/0
```

Note: change the capture network interface (eth0) to the one you are using.

16. Test the basePCap.py script, by capturing the HTTPS (port TCP 443) flows between your PC and multiple browsing servers. Discover your machine IP address (*pc_ipaddr*):

```
python basePCap.py -i eth0 -c <pc_ipaddr> -s 0.0.0.0/0 -t 443
```

17. Test the basePCap.py script, by capturing the TCP/IP packets flows between your PC and YouTube servers during the visualization of videos. Discover your machine IP address (*pc_ipaddr*) and the range of YouTube servers IPv4 addresses for your location (*yt_net*). From UA network, *yt_net* should be (extending the network, and by approximation) 194.210.238.0/24.

```
python basePCap.py –i eth0 –c <pc_ipaddr> –s <yt_net>
```

18. (optional) Consider using the Tshark application to perform the previous captures:

```
tshark –i eth0 'host <pc_ipaddr> and net <net_dst/mask>'
```