INFO 056 – Managing and securing computer networks

Assignment 1 : SNMP monitor (intermediate version)

*By Jean-François Grailet & Raphaël Javaux (**group 5**)*

Introduction

This report describes our intermediate version of the SNMP monitor we were asked to implement. We will first review the general architecture of the monitor, then explain in details how we achieved the discovery of SNMP agents. The last part of this report will review compilation/execution details and some feedback about the first part of this assignment.

Note that we only worked on the discovery part of the monitor so far. Thus, there is no data retrieval or logging, and this version of our monitor only displays messages in console from time to time to notify the detected SNMP agents to the user.

Architecture of the monitor

Inside the src/ folder, you will find 3 packages :

- main
- discovery
- database

The first package, main, gathers the main class of the program and a *Parameters* class used to parse the input parameters of the program and pass them to the other classes of the monitor through a single object.

The discovery package contains all the classes related to the discovery process, which we will explain in the next section of this report.

The last package, database, contains the classes responsible for storing the data of each agent we monitor. For now, we only store the IP of the host computer and the SNMP version, but this part of the monitor should also be responsible for saving the retrieved data and logging in the final version.

During the next steps of the assignment, we may add a 4th package responsible for the data retrieval and maybe a 5th one to deal with traps and write them into log files.

Our implementation of the discovery process

To detect SNMP agents in a range of address, our monitor will simply send PDUs to each host within the range (except for the addresses .0 and .255, of course). Since we want to detect SNMPv3 agents first, the monitor will send a SNMPv3 PDU (with the appropriate authentication/security information) to the host and waits for a response. If he gets a response, then we consider this host is running a SNMPv3 agent. If we do not get anything, we will send a SNMPv2c PDU to detect that version, and if it fails again, we try in the same manner for SNMPv1. If we do not get any response from that host for each tested SNMP version, we consider the host does not run any SNMP agent.

If this solution works well, it features a major drawback : it is extremely slow if we check each host in a sequence, as it takes several minutes to complete. Due to that, we decided to use several Java threads to test hosts in parallel. The discovery part then consists in some "master thread" (implemented by the class *DiscoveryThread*) creating several other threads (implemented with *DiscoveryUnit*), each of them being assigned to a given host and using SNMP4j to send PDUs to that host. Thanks to this solution, what would take plural minutes before now takes less than one or two (in average).

However, we were worried at some point by the number of threads used by the monitor. Indeed, with the example range given in the assignment (139.165.222.0/24), we have to check 254 hosts, which means that the "master thread" will launch 254 different threads at the same time. This should be no big deal for the JVM, if we assume the computer does not have too much running processes. But what if the input address range has a smaller subnet mask (i.e. there are 508, 1016… addresses to test) ? The monitor will launch even more threads, which will be problematic at some point.

This is why the discovery package features a third class, *DiscoveryGuardian*. This class is used to create a single object during execution that will be shared among the "master thread" and the "discovery units" and accessed concurrently. After creating a new unit, the master thread will signal it to the so-called "guardian" so it can increment a number of active units. And before creating the next unit, the master thread checks by the guardian if there is less threads than a given limit (let us say N = 64 units). If there are already N units, it will wait a bit before re-asking. The units themselves will signal to the guardian when they are done, so the counter of active units is decremented to allow the master thread to create a new unit. Thanks to that guardian, the monitor will never use more than N units no matter how big is the number of hosts to check.

To be more flexible, our monitor allows the user to add a 11th parameter upon launch to edit N. If this 11th parameter is missing, it will be set to a default value : 64.

After checking every address in the range, the master thread waits 5s, dumps the currently running SNMP agents and performs the discovery again to update its list of SNMP agents.

Compilation

To compile our code, put the src/ folder and the build.xml file in your working directory and type the command **ant**. This will compile the monitor and produces a build/ folder with the compiled sources along a SNMPMonitor.jar file.

If you need, for some reason, to clean the working directory from our compiled sources, just type **ant clean**. This will delete the .jar as well as the build folder.

BEWARE : the build.xml file assumes that the .jar providing the SNMP4j library is located at /opt/snmp4j-2.2.5/dist, just like on Bee. If you compile our code elsewhere, make sure to edit the build file accordingly.

Execution

To launch the monitor, use the following command :

```
java -jar SNMPMonitor.jar [0] … [9]
```

Where [0] … [9] are the input parameters, in the same order as in the assignment. In other words, if you wish to use the provided parameters, use the following command :

```
java -jar SNMPMonitor.jar /logs 139.165.222.0/24 public v3user authPriv SHA1 stupidAuthPassword AES stupidPrivPassword 7005
```

To edit the number of maximum threads used for discovery, add parameter [10] at the end of the line (must be a positive integer). In other words, if you wish to use at most N = 128 threads (besides the master thread), you will use the following command :

```
java -jar SNMPMonitor.jar /logs 139.165.222.0/24 public v3user authPriv SHA1 stupidAuthPassword AES stupidPrivPassword 7005 128
```

After launch, the monitor will write on the standard output the current configuration, then write a small message each time it discovers/updates an agent. It will also write (on the standard output, once again) the complete list of the agents it detected from time to time.

Feedback

As asked, here is some feedback regarding this intermediate version. First, it took us around 25 hours of work (roughly) to code the monitor as it is now, which includes getting familiar with the SNMP4j library and testing very early solutions (for example, the one sending PDUs to each host in a sequence).

Our major difficulty for that part was getting used to SNMP4j and its tools, after what implementing the Discovery process was a rather interesting software design Homework.