ROCK THE NET

SEW

Isabella Dall O'glio, Astrid Krickl, Christian Maran, Bernhard Schwertberger

26.09.2014

Content

Job description

Trained competencies:

- Using APIs, Network programming
- Application programming: GUI-programming, parallel programming
- software engineering: buildsystems, testing with mock-objects, design patterns

Basic tasks

Implement a simple-to-use application to monitor and configure a hardware firewall appliance "Juniper NetScreen 5GT ". The firewall allows read access over the SNMP-protocol (your app should be able to test if SNMPv3 is available and if not fallback on SNMPv2c) and write access over Telnet.

Your app should accomplish following tasks:

- List all configured firewall rules (policies) on the device, add the details of the mentioned services and zones as well.
- Allow refreshing of the list by clicking a button and by a configurable time-intervall. Your GUI should remain responsive even with short refresh-intervals!
- Visualize the thru-put for a highlighted firewall-rule (nice2have: multiple rows) in a line-chart (configurable refresh-interval, unit bytes/sec)
- Encapsulate the data retrieval for further reuse and easy expansion. An UML-model of your design will help you defend it at the review!
- Build a visual appealing and easy to use interface (there is more than Swing out there).

Additional information:

- Since there is only one firewall-appliance available, the time each team can test with the hardware will be strictly limited. Therefore it is essentially to use mock-objects to allow testing the app during times where the hardware is not available.
- An additional benefit of using mock-objects will be, that a CI-Server can use them for automated building and testing.
- You only need to consider firewall-rules for TCP and UDP connections in IPv4.
- You can find Information about the SNMP-Mibs special for the manufacturer of the used appliance here (maybe not all of the Mibs work with the used model): http://www.oidview.com/mibs/3224/md-3224-1.html
- For exploring the SNMP-Data coming from the appliance you can use tools like this:

  http://ireasoning.com/mibbrowser.shtml

Effort

Estimated working time

Name	Task	Estimated time
Dall' Oglio, Krickl, Maran, Schwertberger	Reading the framework's documentation	4h 00min
Krickl, Maran,	SNMP-Interface implementation	6h 00min
	Display all firewall rules(policies) / Loading data and displaying it in a table	2h 00min
	Refreshing the rules	2h 00min
	Thru-Put Visualization	4h 00min
Dall'Oglio, Schwertberger	GUI: Connect-Page	2h 00min
	GUI: Firewall Monitoring-Page	3h 00min
	Modultesting SNMP-Interface	3h 00min
	Modultesting GUI	1h 30min
	Integrationtesting SNMP-Inface -> GUI	4h 30min
	Systemtest SNMP-Appliance	3h 00min
	Total	**35h 00min**

Required working time

Name	Task	Required time
Dall' Oglio, Krickl, Maran, Schwertberger	Reading the framework's documentation	8h 00min
Krickl, Maran,	SNMP-Interface implementation	1h 00min
	Display all firewall rules(policies) / Loading data and displaying it in a table	4h 00min
	Refreshing the rules	1h 00min
	Thru-Put Visualization	3h 00min
Dall'Oglio, Schwertberger	GUI: Connect-Page	1h 00min
	GUI: Firewall Monitoring-Page	4h 30min
	Modultesting SNMP-Interface	5h 00min
	Integrationtesting SNMP-Inface -> GUI	3h 30min
	Systemtest SNMP-Appliance	2h 00min
	Total	**32h 00min**

Design

For the realization of the tasks we will use Python 3.4.

The backend of the site will be implemented with Flask a Python Web-Framwork where every page is considered as a function. Getting the data from the SNMP-Protocol will be accomplished with the PySNMP framework.

For the frontend Jinja4 will be used to display the data from the Python webservice on the HTML Page. The Traffic-Monitoring could be solved with PyPanl, but we still have to read this framework's documentation. For the GUI-Design we still have to look for some nice look & feel frameworks, like Twitter Bootstrap or Angular.js.For the refreshing of the firewall policies in the GUI we still have to discuss which service we use. By choice we got Server-Sent Events, Websockets, Comet.

Research

Webframework

- Flask
- Blueprints
    - We thought about using Blueprints to split the system up in **more FILES**

snmp-interface

- PySNMP
    - pySNMP is giving us good examples to retrieve the data from the MIB

monitoring

- PyPANL
    - PyPanl already gives us the possiblity to monitor network transfer

refreshing

- Server-Sent Events
- Websockets
    - Quite a volumnious API so we are little bit scared…
- Comet/APE
    - We don't know if all the stuff is necessary to realize this excerise
- Angular.js
    - Easy to use and an interesting way to solve the problem

responsive-gui

- Twitter Bootstrap - http://getbootstrap.com/

Working protocol

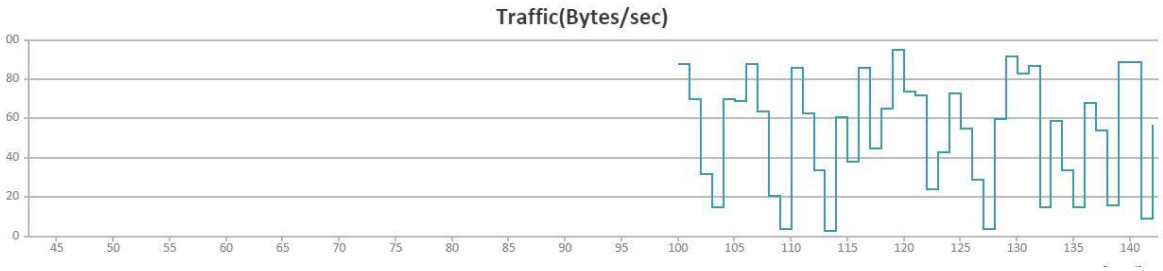
In the school lessons, we mostly did research for the project and designing some prototypes especially for the libraries mentioned before in the design chapter. Since we had some time issues mostly because we couldn't establish a reasonable design concept, we decided to do make a "daily meeting" in the holidays where we met in a coffeehouse and practised some kind of "extreme programming" (because of lack of space).

Used librarys:

- pysnmp
- Flask
- Flask_classy
- json
- random
- canvas.js

Problems

- Time issues
  - Mostly because of the lack of motivation
- PyPanl which we were looking forward to use in our project because it would've fulfilled the visualization task. But what we didn't know at the beginning was that we had to create an account at the Panl homepage to receive a Panl URI. The only problem was that we had to own a credit card for the registration (even for the Trial Version). → Therefor we used canvas.js to draw the diagram which receives the data via python json.



The diagram in the image is filled with random values, since the "nsPlyMonBytePerSec" oid provided only values like(int 0).

Another problem was a misunderstanding in the team about getting the data into the GUI

- Misunderstanding: Displaying the data via Javascript
- Solution: Jinja4

Tests

Unit Tests

Wrote Unit tests based upon the Connection class, which test the Connection and get Methods.

Test cases for succeeding and failing Connection.

This testcase tries to issue a connection with the right parameters.

```
def test_ConnectionRight(self):
    con = Connection
    self.assertTrue(con.checkcon(con, "10.0.100.10", 161, "5xHIT"))
```

This testcase tries to issue a connection with the right parameters except for the IP that fails and returns a False value.

```
def test_ConnectionIPerror(self):
    con = Connection
    self.assertFalse(con.checkcon(con, "123.456.789.10", 161, "5xHIT"))
```

This testcase tries to issue a connection with the right parameters except for the Port that fails and returns a False value.

```
def test_ConnectionPorterror(self):
    con = Connection
    self.assertFalse(con.checkcon(con, "10.0.100.10", 0, "5xHIT"))
```

This testcase tries to issue a connection with the right parameters except for the Groupname that fails and returns a False value.

```
def test_ConnectionGrouperror(self):
    con = Connection
    self.assertFalse(con.checkcon(con, "10.0.100.10", 161, "55555"))
```

Test cases for succeeding and failing get.

This testcase tries to get a rule and checks if it returns something.

```
def test_getright(self):
    con = Connection
    a = con.get(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 161)
    self.assertIsNotNone(a)
```

This testcase tries to get a rule with false parameters in the get method and compares it with a rule that is returned by the get method with right parameters.

```
def test_getfail(self):
    con = Connection
    a = con.get(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 0)
    b = con.get(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 161)
    self.assertNotEqual(a,b)
```

Test cases for succeeding and failing getBulk.

This testcase tries to get a ruleset and checks if it returns something.

```
def test_getBulkright(self):
    con = Connection
    a = con.getbulk(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 161)
    self.assertIsNotNone(a)
```

This testcase tries to get a ruleset with false parameters in the get method and compares it with a ruleset that is returned by the get method with right parameters.

```
def test_getBulkfail(self):
    con = Connection
    a = con.getbulk(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 0)
    b = con.getbulk(con, '1.3.6.1.4.1.3224.10.1', '5xHIT', "10.0.100.10", 161)
    self.assertNotEqual(a,b)
```

Starting the project

Option 1:

Double Click the Run.py file and open your browser with the url: 127.0.0.1:5000

Option 2:

Use these commands in your command shell:

1. pip install virtualenv
2. virtualenv venv
3. venv\Scripts\activate
4. pip install –r requirements.txt
5. cd src
6. python Run.py

Sources

http://www.w3schools.com/html/html5_serversentevents.asp

http://flask.pocoo.org/

http://flask.pocoo.org/docs/0.10/blueprints/

http://pysnmp.sourceforge.net/

http://blog.panl.com/2011/09/13/using-panl-to-monitor-snmp-enabled-network-devices/

https://ws4py.readthedocs.org/en/latest/

http://ape-project.org/

https://angularjs.org/