Adding new equipments

Introduction:

This document is intended to explain how to add a new equipment to the Springbok project. It will explain:

- How is structured the project
- Which tools are used for parsing and where to add the new equipment parser

This guide will show one solution with examples on how to add an equipment. You are free to use a different architecture for your parser and you are really encourage to have a look at the other equipments and the source code for better comprehension.

Update:

- Since you can have more than one firewall per configuration file, the function get_firewall return a list of firewall
- To support Iptables and its action (chained model), the class Action was added

Table of Contents

Structure of the project

To build a parser it is necessary to know the following classes of project:

- Firewall
- Interface
- ACL
- Rule
- Operator
- Protocol
- Ip
- Port
- Action
- NetworkGraph (singleton)

Firewall

The firewall class represent a firewall. It owns the following attributes:

- interfaces (**Interface list**, list of Interface)
- acl (**ACL list**, list of ACL)
- type (**string**, representing the firewall type)
- name (**string**, file path of the configuration file)
- hostname (**string**, firewall hostname)
- unused_objects (**set** of unused objects)
- unbounded_rules (**set** of unbounded rules)
- dictionnary (**dictionary** of defined objects)

Interface

The interface class represent an interface. It owns the following attributes:

- nameif (**string**, name interface)
- network (**Ip**, address of the interface)
- name (**string**, name of the interface)
- sub_interfaces (**list**, sub interfaces)
- attributes (**dictionary** of attributes ex: tag values)

ACL

The ACL class represent an ACL. It owns the following attributes:

- name (**string**, acl name)
- rules (**Rule list**, list of rules of the acl)

Rule

The Rule class represent a rule. It owns the following attributes:

- identifier (**int**, identifier number)
- name (**string**, rule name)
- protocol (**Operator list**, protocol)
- protocol_name (**string**, protocol name)
- ip_source (**Operator list**, ip source)
- ip_source_name (**string**, ip source name)
- port_source (**Operator list**, port source)
- port_source_name (**string**, port source name)
- ip_dest (**Operator list**, ip destination)
- ip_dest_name (**string**, ip destination name)
- port_dest (**Operator list**, port destination)
- port_dest_name (**string**, port destination name)
- action (**bool**, action True for permit, False for deny)

Operator

The Operator class represent an operator. It owns the following attributes:

- operator (string, operator type 'EQ', 'LT', 'GT', 'NEQ', 'RANGE')
- v1 (**Protocol/Ip/Port**, value of the operator)
- v2 (**Protocol/Ip/Port**, used for range)

Protocol

The Protocol class represent a protocol. It owns the following attributes:

- protocol (**int**, protocol value)

Ip

The Ip class represent an ip address with its mask. It owns the following attributes:

- ip (int, ip value)
- mask (**int**, mask value)

Port

The Port class represent a port. It owns the following attributes:

- port (**int**, port value)

Action

The action class represent an action. It owns the following attributes:

- chain (**ACL, Bool or string**, the action)
- goto (**Bool,** see iptables goto action)

The chain argument is:

- True if the rule is accepted
- False if the rule is rejected
- An ACL if the rule is chained
- A string for any other special action

NetworkGraph

The NetworkGraph class is a singleton used to create and represent the network graph. NetworkGraph contain the function:

- bind_acl(acl, firewall, ip1, ip2), bind the acl between ip1 and ip2 to the firewall.
  - ip1 and ip2 can be an Interface or a Firewall

Summary of the project structure and diagram class

To create new firewalls you need to

- Add all necessary element to the firewall (hostname, name, unused_objects, …)
- Detect interface, create them and add them to the firewall
- Detect ACL, create them and add them to the firewall
- Bind ACL to the NetworkGraph with the 'bind_acl' function
- Detect and create rules and them to the corresponding ACL

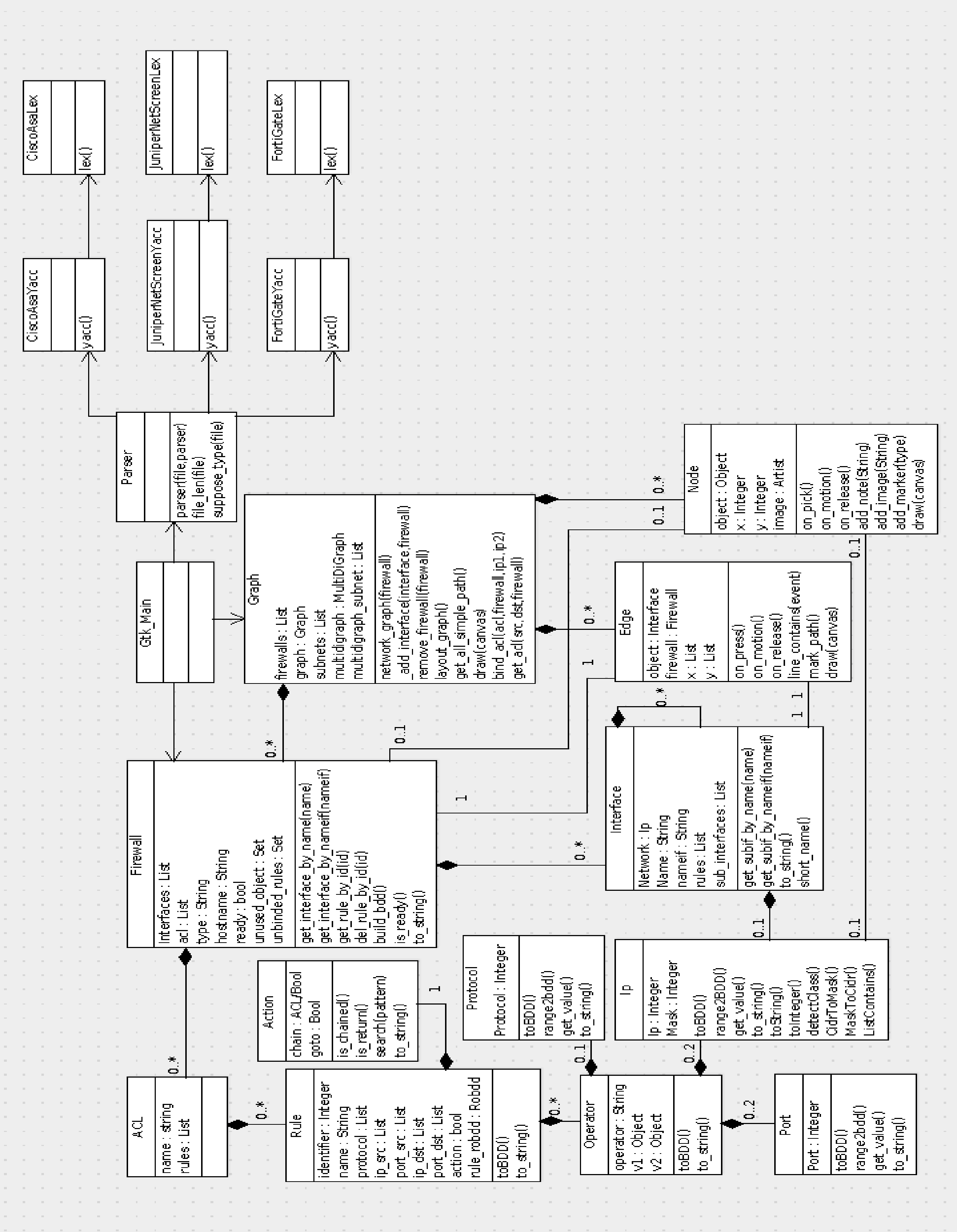
Note on rule construction

- Rules are of the form:
  - 'Id' 'Name' 'Protocol' 'Ip_source' 'Port_source' 'Ip_destination' 'Port_destination' 'Action'
  - To reduce memory consumption we use operators to represent Protocol, Ip source and destination and Port source and destination. Protocol, Ip and Port field are a **list of operators** (each field can have a bunch of values). An operator is an instance class of

Operator with v1 and v2 as instance class of Protocol, Ip or Port. An operator posses the following operation:

- 'LT' → lower than
- 'GT' → greater than
- EQ → equal
- NEQ → not equal
- RANGE → range (specify v2 in this case)

Diagram class



Parsing a new equipment

Tool used to parse configuration file

Springbok use Python Lex Yacc (PLY) to parse configuration file equipments. You must be a little familiar with Lex Yacc before continuing because this document will not explain in details how PLY work. For more information you can refer to : http://www.dabeaz.com/ply/

Parsing a new equipments

Creating folder and files

- Create a folder in the folder Parser/ for your new equipment (ex: add folder CiscoAsa/ in Parser/).

- To parse a new equipment, you need two files:

- The Lex file for the tokens (ex: CiscoAsaLex.py)
- The Yacc file for the grammar (ex: CiscoAsaYacc)

Add the Lex and the Yacc file to your new folder.

Fill the lexer

- Import the lexer and the regex module:

```
import re
from Parser.ply import lex
```

- Define your tokens (note the presence of '$' at each end of token in 'reserved'):

```
reserved = {
    r'any$|any4$': 'ANY',
    r'accept$|permit$': 'ACCEPT',
    r'deny$|reject$': 'DENY',
    r'access-group$': 'ACCESS_GROUP',
    [...]
    r'neq$': 'OP_NEQ',
    r'range$': 'OP_RANGE',
}

tokens = [
             'BANG',
             'IP_ADDR',
             'NUMBER',
             'WS',
             'NL',
             'WORD',
         ] + list(reserved.values())
[...]
def t_IP_ADDR(t):
    r'\d+\.\d+\.\d+\.\d+'
    return t
[...]
```

- In most lexers in this project you will find the token WORD. WORD is intended to be a wildcard to match words, variables, …. To do this and avoid to match reserved token, you must verify you are not matching a reserved word:

```
def t_WORD(t):
    r'[a-zA-Z0-9/\\\.,_-]+'
    # Check for reserved words
    for k, v in reserved.items():
        if re.match(k, t.value, re.I):
            t.type = v
    return t
```

- To ignore white space:

```
def t_WS(t):
    r'[ \t]+'
    pass
```

- Define your lexer :

```
lexer = lex.lex()
```

Fill the Parser

- Import the parser, your tokens and your lexer:

```
from Parser.ply import yacc
from Parser.CiscoAsa.CiscoAsaLex import tokens
from Parser.CiscoAsa.CiscoAsaLex import lexer
```

- Import project class for construction:

```
from SpringBase.Ip import Ip
from SpringBase.Protocol import Protocol
from SpringBase.Port import Port
from SpringBase.Interface import Interface
from SpringBase.Rule import Rule
from SpringBase.Operator import Operator
from SpringBase.Firewall import Firewall
from SpringBase.ACL import ACL
import NetworkGraph
```

- In most parser in this project, you will find an object_dict dictionary structure. This is used to remember defined objects.

```
object_dict = {}
```

- The object_dict is a dictionary of list of dictionaries:

```
object_dict[p_info['object_name']].append({'network': Operator('EQ', Ip(p[3]))})
```

- In most parser in this project, you will find a p_info dictionary structure. This is used to remember informations and state of the parser.

```
p_info = {
    'firewall': Firewall(),
        'interface_state': False,
        'current_interface': None,
        [...]
        'raise_on_error': False,
    }
```

- To function properly, you **must** implement at least the following functions:

- init: this function is called once, before parsing the configuration file. You can use it to instantiated your structures and begin to fill the firewall informations.
    - name is the file name
    - raise_on_error is a boolean used to raise an error if the parsing fail (see: p_error)

```
    def init(name, raise_on_error=False):
        object_dict.clear()
        p_info['firewall'] = Firewall()
        p_info['firewall'].name = name
        p_info['firewall'].hostname = ntpath.basename(name)
        p_info['firewall'].type = 'CiscoAsa'
        p_info['interface_state'] = False
        p_info['current_interface'] = None
        [...]
```

- update: this function is called after each parsed line. You can use it to reset a state or update informations.

```
    def update():
        p_info['current_rule'] = Rule(None, None, [], [], [], [], [], False)
        p_info['index_rule'] = len(p_info['rule_list']).
```

- finish: this function is called once, after finishing parsing the configuration file. You can use it to complete your firewall.

```
    def finish():
        # add dictionary to firewall
        p_info['firewall'].dictionnary = dict(object_dict)

        # perform unused object
    for k in object_dict:
        if k not in p_info['used_object']:
            p_info['firewall'].unused_objects.add(k)
```

- get_firewall: this functions is called once, after the finish function. This is used to return the list of built firewalls.

```
def get_firewall():
    return p_info['firewall']
```

- Implement the equipment grammar. You need at least to parse the following elements:

- variables objects

```
def p_object_line_2(p):
    '''object_line : OBJECT NETWORK item RENAME item
                   | OBJECT SERVICE item RENAME item'''
    dict[p[5]] = object_dict.pop(p[3])
    p_info['object_name'] = p[5]
[...]
### network_line
def p_network_line_1(p):
    '''network_line : HOST IP_ADDR'''
    object_dict[p_info['object_name']].append({'network': Operator('EQ', Ip(p[3]))})
[...]
```

- interfaces

```
### interface_line
def p_interface_line(p):
    '''interface_line : INTERFACE item
                      | INTERFACE REDUNDANT item
                      | INTERFACE PORT_CHANNEL item
                      | BANG'''
    if p[1] == '!':
        p_info['interface_state'] = False
    else:
        [...]
```

```
# interface name parse
### interface_name_line
def p_interface_name_line(p):
    '''interface_name_line : NAMEIF item'''
    p_info['current_interface'].name = p[2]

# ip address parse
### ip_address_line
def p_ip_address_line_1(p):
    '''ip_address_line : IP ADDRESS IP_ADDR ip_address_option'''
    p_info['current_interface'].network = Ip(p[3], None, True)
[...]
```

- hostname

```
### hostname line
def p_hostname_line(p):
    '''hostname_line : HOSTNAME item'''
    p_info['firewall'].hostname = p[2]
```

- ACL

```
### access_line
def p_access_line_1(p):
    '''access_line : ACCESS_LIST item line_number EXTENDED rule
                   | ACCESS_LIST item STANDARD line_number standard_rule'''
    p_info['current_rule'].identifier = p_info['rule_id']
    p_info['rule_id'] += 1
    p_info['current_rule'].name = p[2]
    p_info['rule_list'].insert(p_info['index_rule'], p_info['current_rule'])
[...]
```

- **Bind ACL** to the network (in this example we use Cisco Asa, so we bind on access-group line)

```
### access_group_line
def p_access_group_line_1(p):
    '''access_group_line : ACCESS_GROUP item IN INTERFACE item optitem
                         | ACCESS_GROUP item OUT INTERFACE item optitem'''
    interface = p_info['firewall'].get_interface_by_name(p[5])
    firewall = p_info['firewall']
    acl = firewall.get_acl_by_name(p[2])
    if not acl:
        acl = ACL(p[2])
        p_info['firewall'].acl.append(acl)
        p_info['bounded_rules'].add(p[2])
        NetworkGraph.NetworkGraph.NetworkGraph().bind_acl(acl, firewall, interface, firewall)
        NetworkGraph.NetworkGraph.NetworkGraph().bind_acl(acl, firewall, firewall, interface)
```

- You will find the CiscoAsa grammar used to construct the firewall in the annexes section (take also a look at the existing parser in the project).

- Add a p_error rule for errors:

```
def p_error(p):
    if p:
        print("Syntax error at '%s'" % p.value)
    else:
        print("Syntax error at EOF")

    if p_info['raise_on_error']:
        raise SyntaxError
```

- Define the variable parser

```
parser = yacc.yacc(optimize=1)
```

Annexes

Cisco Asa grammar

```
precedence = (
    ('left', 'OBJECT_GROUP'),
)

lines : line
      | line lines

line : access_line NL
     | hostname_line NL
     | access_group_line NL
     | interface_line NL
     | ip_address_line NL
     | interface_name_line NL
     | object_line NL
     | network_line NL
     | service_line NL
     | object_group_line NL
     | group_object_line NL
     | icmp_object_line NL
     | network_object_line NL
     | protocol_object_line NL
     | port_object_line NL
     | service_object_line NL
     | words NL
     | error NL
     | NL

empty :

item : WORD
     | NUMBER

optitem : item
        | empty


words : WORD
      | WORD words


object_line : OBJECT NETWORK item
            | OBJECT SERVICE item
            | OBJECT NETWORK item RENAME item
            | OBJECT SERVICE item RENAME item

network_line : HOST IP_ADDR
             | NETWORK IP_ADDR
             | OP_RANGE IP_ADDR IP_ADDR

service_line : SERVICE item
             | SERVICE tcp_udp opt_service
             | SERVICE ICMP optitem
             | SERVICE ICMP6 optitem

opt_service : SOURCE operator
            | DESTINATION operator
            | SOURCE operator DESTINATION operator
            | empty

object_group_line : OBJECT_GROUP PROTOCOL item
                  | OBJECT_GROUP NETWORK item
                  | OBJECT_GROUP ICMP_TYPE item
                  | OBJECT_GROUP SECURITY item
                  | OBJECT_GROUP USER item
                  | OBJECT_GROUP SERVICE item object_opt_tcp_udp

group_object_line : GROUP_OBJECT item

icmp_object_line : ICMP_OBJECT item

network_object_line : NETWORK_OBJECT HOST IP_ADDR
                    | NETWORK_OBJECT IP_ADDR IP_ADDR
                    | NETWORK_OBJECT OBJECT item

protocol_object_line : PROTOCOL_OBJECT item

port_object_line : PORT_OBJECT OP_EQ item
                 | PORT_OBJECT OP_RANGE NUMBER NUMBER

service_object_line : SERVICE_OBJECT item
                    | SERVICE_OBJECT object_tcp_udp opt_service
                    | SERVICE_OBJECT ICMP optitem
                    | SERVICE_OBJECT ICMP6 optitem
                    | SERVICE_OBJECT OBJECT item

object_tcp_udp : TCP
               | UDP
               | TCP_UDP

object_opt_tcp_udp : TCP
                   | UDP
                   | TCP_UDP
                   | empty

interface_line : INTERFACE item
               | INTERFACE REDUNDANT item
               | INTERFACE PORT_CHANNEL item
               | BANG

interface_name_line : NAMEIF item

ip_address_line : IP ADDRESS IP_ADDR ip_address_option
                | IP ADDRESS IP_ADDR IP_ADDR ip_address_option
                | NO IP ADDRESS optitem

ip_address_option : STANDBY IP_ADDR
                  | CLUSTER_POOL item
                  | empty

hostname_line : HOSTNAME item

access_group_line : ACCESS_GROUP item IN INTERFACE item optitem
                  | ACCESS_GROUP item OUT INTERFACE item optitem
                  | ACCESS_GROUP item optitem

access_line : ACCESS_LIST item line_number EXTENDED rule
            | ACCESS_LIST item STANDARD line_number standard_rule
            | ACCESS_LIST item line_number REMARK words

line_number : WORD NUMBER
            | empty

rule : action protocol user_arg security_arg address_source security_arg address_dest log access_option
     | action tcp_udp user_arg security_arg address_source port_source security_arg address_dest
port_dest log access_option
     | action ICMP user_arg security_arg address_source security_arg address_dest icmp_arg log
access_option

standard_rule : action ANY
              | action HOST IP_ADDR
              | action IP_ADDR IP_ADDR

user_arg : OBJECT_GROUP_USER item
         | USER item
         | USER ANY
         | USER NONE
         | USER_GROUP item
         | empty

security_arg : OBJECT_GROUP_SECURITY item
             | SECURITY_GROUP NAME item
             | SECURITY_GROUP TAG item
             | empty

log : LOG
    | LOG item
    | LOG INTERVAL item
    | LOG item INTERVAL item
    | LOG DISABLE
    | LOG DEFAULT
    | empty

access_option : INACTIVE
              | TIME_RANGE item
              | empty

action : ACCEPT
       | DENY

tcp_udp : TCP
        | UDP

protocol : item
         | IP
         | OBJECT_GROUP item
         | OBJECT item

address_source : HOST IP_ADDR
               | IP_ADDR IP_ADDR
               | ANY
               | INTERFACE
               | OBJECT_GROUP item
               | OBJECT item

address_dest : HOST IP_ADDR
             | IP_ADDR IP_ADDR
             | ANY
             | INTERFACE
             | OBJECT_GROUP item
             | OBJECT item

port_source : operator
            | OBJECT_GROUP item
            | empty

port_dest : operator
          | OBJECT_GROUP item
          | empty

icmp_arg : item optitem
         | OBJECT_GROUP item
         | empty

operator : OP_LT port_service
         | OP_GT port_service
         | OP_EQ port_service
         | OP_NEQ port_service
         | OP_RANGE port_service port_service

port_service : WORD
             | NUMBER
```