Firewall Simulator Documentation

Release 0.1

Leighton Lilford

Sep 09, 2017

CONTENTS:

CHAPTER

ONE

SOURCE

1.1 Socket Webserver

Responses will always be in the form of an array/list.
The first element will be either **"S"** or **"E"** - dictating whether the function completed Successfully, or threw an Error.

1.1.1 General Functions

`webserver.`**`connect`**(*sid*, *environ*)
Used to connect to the socket

Args: sid (str): Unique identifier for the socket connection

The *sid* is automatically assigned, and, assuming you're using a good library, should be automatically sent with all future requests. *sid* is an arg for all requests

`webserver.`**`disconnect`**(*sid*)
Used to gracefully disconnect from the client

1.1.2 Node Functions

`webserver.`**`create_node`**(*sid*, *data*)
Used to create a new node on the network

Args: data (str): A string with a unique node id

Example Request:

```
socket.emit('create-node', <node_id>)
socket.emit('create-node', 0)
```

Returns: JSON Object: `["S", "Node Created"]`

Raises: ValueError: If node id is already taken

`webserver.`**`delete_node`**(*sid*, *data*)
Used to delete an existing node on the network

Args: data (str): A string with the node id

Example Request:

```
socket.emit('delete-node', <node_id>)
socket.emit('delete-node', 0)
```

Returns: JSON Object: `["S", "Node deleted"]`

Raises: KeyError: if node id does not exist

`webserver.`**`connect_nodes`**(*sid*, *data*)
DEPRECATED

`webserver.`**`update_status_table_def`**(*sid*, *data*)
Used to get the current nodes

Args: data (str): Can be either `""` or `"loud"`. `"loud"` will cause the response to be "Table update triggered".

Example Request:

```
socket.emit('update-status-table', null)
socket.emit('update-status-table', "loud")
```

Returns: JSON Object: `["N"]` or `["S", "Table update triggered"]`

This function doesn't return anything, rather triggers a seperate update.
It will trigger the *webserver.update_status_table* function.

1.1.3 Firewall Functions

`webserver.`**`get_firewall`**(*sid*, *node_id*)
Returns the current firewall for a given node id

Args: node_id (str): A string with the node id

Example Request:

```
socket.emit('delete-node', <node_id>)
socket.emit('get-firewall', 0)
```

Returns: JSON Object: `["S", "", <firewall_structure>]` The firewall structure is a list of chains, each of which contains a list of rules [under the `children` key]

Example firewall_structure:

```
{
    [
        {
            "id":"INPUT",
            "label":"INPUT",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j DROP"
                }
            ]
        },
        {
            "id":"FORWARD",
            "label":"FORWARD",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j DROP"
                }
```

```
            ]
        },
        {
            "id":"OUTPUT",
            "label":"OUTPUT",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j DROP"
                }
            ]
        },
        {
            "id":"ACCEPT",
            "label":"ACCEPT",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j ACCEPT"
                }
            ]
        },
        {
            "id":"REJECT",
            "label":"REJECT",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j REJECT"
                }
            ]
        },
        {
            "id":"DROP",
            "label":"DROP",
            "children":[
                {
                    "id":0,
                    "label":"-i Any -o Any -p Any -s Any -d Any -j DROP"
                }
            ]
        }
    ]
}
```

Raises: KeyError: if node id does not exist

`webserver.`**`add_rule`**(*sid*, *data*)

Adds rule to a node's firewall.

Args: data (JSON): `[node_id, chain_id, <rule_object>]`

Example Request:

```
{
    [
        0,
        "INPUT",
```

```
        {
            "chain":"New_Chain",
            "dst":false,
            "src":"",
            "input_device":"",
            "output_device":false,
            "protocol":false
        }
    ]
}
```

Returns: `["S","Rule added."]` or `["S","Rule added. New chain New_Chain created"]`

In the above, "INPUT" is the chain the rule is appended to, and "New_Chain" is the name of the chain the rule processor goes to if the rule matches the given packet.

Note that "New_Chain" can be either a new chain, an existing chain, or a final status ["ACCEPT"; "DROP"; "REJECT"]

`webserver.`**`delete_rule`**(*sid*, *data*)

Delete a set of rules from a node's firewall

Args: data (JSON):

```
{
    [node_id,
        [
            [chain1_id, rule1_id],
            [chain2_id, rule2_id],
            ...
        ]
    ]
}
```

Example Request data:

```
{
    [0,
        [
            ["INPUT",0]
        ]
    ]
}
```

Returns: JSON Object: `["S", "Rules deleted"]`

`webserver.`**`update_status_table`**(*sid*)

Send a status update for all nodes

Returns: List of nodes, with their properties

Example Response:

```
{
    [
        {
            "Node_ID":0,
            "Node_Addr":"10.62.0.0",
            "Node_Mac":"62:f4:e5:9c:00:00",
            "Packets_In":"",
            "Packets_Out":""
        },
        {
            "Node_ID":1,
            "Node_Addr":"10.51.0.1",
            "Node_Mac":"1e:68:26:a7:00:01",
            "Packets_In":"",
            "Packets_Out":""
        }
    ]
}
```

Note:

`Packets_In` and `Packets_Out` will be blank when running this function.
The fields are populated after the simulation runs.

1.1.4 Simulation Functions

`webserver.`**`download_sim_file`**(*sid*, *data*)

Used to download a simulation template, in csv format

Args: data (str): Just leave empty

Returns: Simulation template `["S", "", <template>]`

Example Response template: `"packet_id,network_layer,application_layer, source_port,destination_port,source_ip,destination_ip,input_device, output_device,ttl \r\n 1,icmp,,,,10.47.0.0,,eth1,eth1,2 \r\n"`

`webserver.`**`upload_simulation_file`**(*sid*, *data*)

Upload the packets to be simulated. This should be called before running the simulation.

Args: data: (str): The simulation file [csv format]

Example Request data:: `"packet_id,network_layer,application_layer,source_port, destination_port,source_ip,destination_ip,input_device,output_device, ttl\r\n1,icmp,,,,10.47.0.0,10.48.0.0,eth1,eth1,2"`

Returns: `["S","Simulation file uploaded"]`

Raises: If a row is invalid you will get an error stating which row failed

Note: The first row is ignored [as it contains column titles]

`webserver.`**`run_simulation`**(*sid*, *data*)

Runs the simulation on the created nodes with the uploaded packets.

Args: data (str): Leave empty

Returns: `["S","Simulation Complete"]`

Note:

Packets **must** have been set from the *webserver.upload_simulation_file* method.
Results can be retrieved from the *webserver.get_sim_results* method

webserver.**get_sim_results**(*sid*, *data*)

Returns the simulation results after running. Results are in 3 parts: packet results; node results and rule results

- Packet results: 1 row per packet, stating what happened to it [dropped; accepted...]
- Node results: 1 row per node per packet. States what happened to a packet at each node it reached
- Rule results: 1 row per rule per packet. States what happened to a packet at each rule in each node it hit

Args: data (str): Leave empty

Returns: ["S","", <results>]

Example results:

```
{
    "packet":"Packet_ID,Source_IP,Destination_IP,Protocol,Result\r\n-1,10.87.0.0,
↪10.198.0.1,ICMP,DROP\r\n-1,10.198.0.1,10.87.0.0,ICMP,DROP\r\n",
    "node":"Packet_ID,Hop_Number,Node_IP,Direction,Protocol,Result\r\n-1,1,10.87.
↪0.0,Output,ICMP,DROP\r\n-1,1,10.198.0.1,Output,ICMP,DROP\r\n",
    "rule":"Packet_ID,Node_IP,Chain,Protocol,Rule,Result\r\n-1,10.87.0.0,OUTPUT,
↪ICMP,P:ICMP S: D: iD: oD:,DROP\r\n-1,10.198.0.1,OUTPUT,ICMP,P:ICMP S: D: iD:␣
↪oD:,DROP\r\n"
}
```

Note: This must only be called **after** running the simulation - see *webserver.run_simulation*

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

INDEX