Developer's Guide

Lyndon Fawcett
l.fawcett1@lancaster.ac.uk
or
lyndonfawcett@gmail.com

April 9, 2018

Executive Summary

This document is aimed at developers for TENNISON, reducing the learning curve of execute, testing, using, and adding to the system. Enclosed is a comprehensive developer's guide on how to use and build on top of the TENNISON system. It firstly explains a technical overview of the system, describing relationships between system components and their associated code. Supporting this, snippets from the Git wiki are included in the appendix. After this, it covers a "getting started" guide, explaining how to get the system up and running, including how to build the latest version of ONOS with the TENNISON applications. Finally, after understanding TENNISON, the guide shows one how to exercise the system through a series of attacks.

Please note that this document is a continuing work in progress and is not necessarily up to date with the codebase.

Contents

1 Technical system overview

This section covers the technical aspects of the system, firstly describing the system as a whole, then moving into detail on specific components and APIs.

The general relationship between the various TENNISON components and their code are explained in Appendix A. These have been split into 4 sections, from the bottom up, this includes: the network, the network controller (ONOS), the coordinator, and northbound applications. In the example, all northbound applications are written in python, though as they are interfaced over rest, new applications could be written in any language[1]. ONOS's northbound applications are written in Java using the Karaf framework Maven build files to perform linking between interfaces and primary implementations of interfaces or abstract classes. The coordinator and the truffle code around snort (not shown on diagram) are both written in python 3.

Appendix B shows the class diagram of the coordinator. A python application that holds a set of thresholds that affect the southbound connection to the network via ONOS and are modified by the northbound interface through remote applications.

Appendix C describes the specific details around the ONOS Flow API. These are otherwise known as onos-tennison-apps in the git project. The REST API here allows one to modify the network, redirecting and blocking traffic. Later in this appendix specifics around implementations of intents are discussed.

API calls offered by TENNISON's northbound interface (known as watson.py in code) are detailed in Appendix D. This also instructs one on how to install an application such that it can be started, stopped, and altered by the coordinator (and the GUI).

For modification of TENNISON's external components one should refer to the collectors. These are shown in Appendix E and describe the communication generated from snort, sFlowRT, and ONOS-IPFIX to the coordinator. A new one of these collectors would have to be created to support any additional monitoring engines or DPIs.

Appendix F describes the REST calls available for modifying rules within snort. The live manipulation of snort rules is managed in the tennison.py class (not shown in the relationship diagram).

[1]Note: if an auto-ran then command would have to be modified to match language, the default is python.

2 TENNISON usage guide

2.1 Getting started (The quick way)

The following guide is assuming you are using vagrant box 11 with the correct vagrant file. It is recommended that this is ran with at least 2GBs of RAM, and more if running at scale. Currently there are various steps required to do to ensure that the system operates as expected.

To launch ONOS locally[2]:

```
$ onos-buck run onos-local clean
in /home/vagrant/onos/
```

Using the keyword "clean" in the command makes sure that the the local instance of ONOS will be refreshed. Running as a service will run an older version and will not work.

To install the ONOS apps:

```
$ ./install_apps_remote
in /home/vagrant/secapp/onos-tennison-apps
```

These apps must be installed each time the ONOS instance is reset.

To launch Mininet with TENNISON and snort:

```
$ sudo ./tennison_dev.py -m -a -p
in /home/vagrant/secapp/topology/tennison_dev/
```

To access the ONOS GUI go to:

```
http://127.0.0.1:8181/onos/ui/index.html
```

To access the TENNISON GUI go to:

```
http://127.0.0.1:8080/
```

2.1.1 TENNISON experimenter

This is the latest tool for running TENNISON and clustered TENNISON. Check this tool out in the tools/dev/ directory.

[2]If ONOS is restarted then Mininet should also be restarted, otherwise ONOS may not detect the Snort instances.

2.2 Getting started (From scratch)

Before reading this section, see install_tennison.sh in the coordinator git repository

2.3 Developer guide

This is a quick guide on how to develop with ONOS inside our vagrant box. ONOS provides various guides on how to do this, however they can be misleading as the documentation is fragmented and based on different versions of the code. https://wiki.onosproject.org/display/ONOS/Developer+Guide

To build ONOS (assuming version 1.10 or higher is used) do:

```
$ onos-buck build onos
in /home/vagrant/onos/
This should take around 5-10 minutes to run, less time if rebuilding.
```

To build the ONOS apps:

```
$ onos-buck-publish-local
```

```
$ mcis
in /home/vagrant/secapp/onos-tennison-apps
```

To install ONOS apps run (ONOS must be running):

```
$ ./install_apps_remote
in /home/vagrant/secapp/onos-tennison-apps
To make sure the apps are launched in the right order, restart ONOS
```

Instructions on how to install used an IDE with ONOS can be found at: https://wiki.onosproject.org/display/ONOS/Importing+ONOS+projects+into+IntelliJ+IDEA

2.3.1 Upgrading the ONOS version

First, copy the SecurityPipeline.java file, the onos-drivers.xml file and the local cell file from the current ONOS installation:

```
$ cp ~/onos/drivers/default/src/main/java/org/onosproject/driver/
pipeline/SecurityPipeline.java ~
$ cp ~/onos/drivers/default/src/main/resources/onos-drivers.xml ~
$ cp ~/onos/tools/test/cells/local ~
```

onos-drivers.xml is used to link together pipelines and devices, it must be modified if testing with something other than OvS. The local cell file tells ONOS the IP address to use when running locally.

Next, remove the old version of ONOS and use git to clone the desired version of the repository from https://github.com/opennetworkinglab/onos :

```
$ sudo rm -r ~/onos
$ git clone https://github.com/opennetworkinglab/onos
in /home/vagrant/
```

Finally, replace the files in the new installation with the files we copied from the old installation, source the bash profile and build ONOS:

```
$ mv ~/SecurityPipeline.java ~/onos/drivers/default/src/main/java/org/onospro
pipeline/
$ rm ~/onos/drivers/default/src/main/resources/onos-drivers.xml
$ mv ~/onos-drivers.xml ~/onos/drivers/default/src/main/resources/
$ rm ~/onos/tools/test/cells/local
$ mv ~/local ~/onos/tools/test/cells/
$ source ~/onos/tools/dev/bash_profile
$ onos-buck build onos
```

This may take some time to complete as ONOS we have to rebuild from scratch.

To update the ONOS apps, look in the /home/vagrant/onos/onos.defs file for "ONOS VERSION", then find and replace the version (assuming ONOS 1.10.4 was the old version) and rebuild the apps using the commands:

```
$ grep -rl '1.10.4-SNAPSHOT' ./ | xargs sed -i
's/1.10.4-SNAPSHOT/{ONOS_VERSION}/g'
$ mcis
in /home/vagrant/secapp/onos-tennison-apps/
```

If the ONOS version is relatively new, the maven repositories may not have been updated yet so the additional command below must be run before building the apps:

```
$ onos-buck-publish-local
in /home/vagrant/onos
```

Manually update onos-drivers.xml. Copy over the line for the security driver to the latest onos-drivers.xml

3 Attack demonstration guide

This section includes attack instructions and explanations which were used for the WP4 demonstration.

3.1 Attack Demo - Port scan

Firstly, enable the ipfix portscan app in the TENNISON GUI.

Exploit Portscan attack on single host (h12)

```
mininet> h8 nmap -n -p- -sA 10.0.0.12
```

Note: -sA (SYN scan) instead of -sT (connect() scan). A connect() scan causes the source port to also change and an actual connection is attempted, this results in a port scan being detected in both directions and therefore both being blocked. -n to prevent nmap resolving addresses (delay at the start)

Detection The ipfix-portscan-app monitors the variance of ports being contacted by a single host. On the detection of an ongoing port scan from a specific source the following ipfix threshold is installed:

```
"subtype": "prefix",
"treatment": "block",
"fields":{"sourceIPv4Address": X.X.X.X},
"treatment_fields":{"sourceIPv4Address": X.X.X.X},
"priority": 10
```

3.2 Attack Demo - DDoS

Firstly, enable the ipfix DDoS app in the TENNISON GUI.

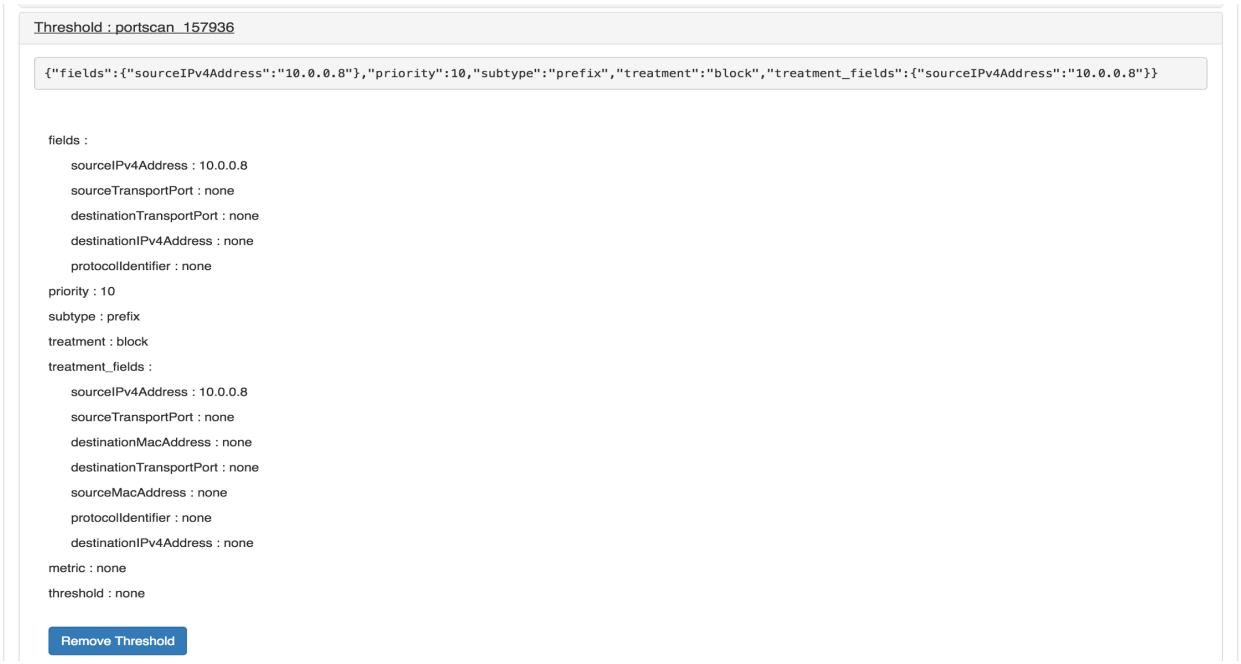


Figure 1: Port scan threshold details

Exploit Running SYN flood DDoS attack on a web server (h15). The distributed aspect is simulated by spoofing random source IP addresses (–rand-source).

```
mininet> h10 hping3 -c 500 -d 120 -S -w 64 -p 80 --fast --rand-source h15
```

For sending at different rates use '-i' and specify the delay between packets.

Detection The distributed nature is first picked up by ipfix-ddos-app which discovers nodes with a high variance of querying nodes. This is then considered a server undergoing a potential DDoS attack and so all traffic querying the server is mirrored by ipfix-ddos-app installing the following (example) threshold:

```
'treatment_fields': {'destinationIPv4Address': '<server>'},
'treatment': 'snort_mirror',
'fields': {'destinationIPv4Address': '<server>'},
'subtype': 'prefix',
'priority': 10
```

The idea what then to use snort to make the distinction between malicious traffic and legitimate traffic, which may not be possible. The extent of snort rules for detecting SYN flood attack only goes as far as:

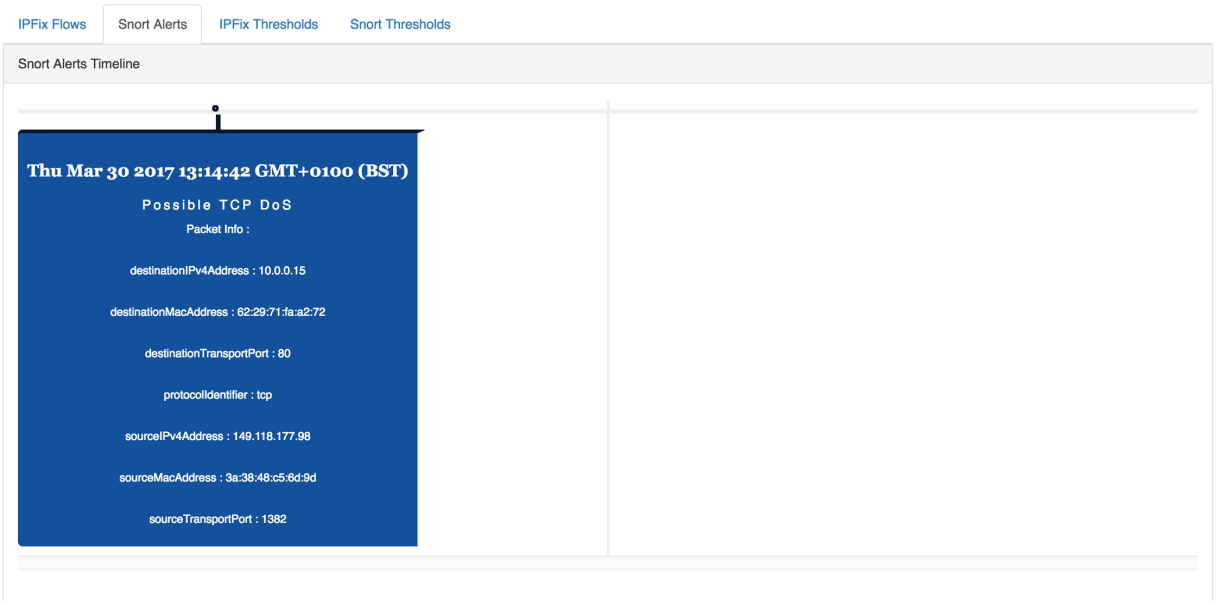


Figure 2: Screen capture of GUI displaying snort DDoS alert

Other Solutions

1. It may be possible to differentiate by some other payload value (insert something unique to the attack traffic).
2. Different DDoS attacks may be possible for snort to separate attack traffic from legitimate triaffic .e.g jolt attack, teardrop, IGMP, dos auth - these all have corresponding rules in dos.rules - after looking into there they all seem specific to windows NT/95 etc.
3. Remediate on the snort alert, would cause all traffic to be blocked and therefore enforcing the denial of service. The following snort threshold would be used to match the snort alert and then block the destination IP address.

3.3 Attack Demo - Intrusion detection

No app is required for detection of this exploit (though one could be made to enhance detection).

Exploit To demonstrate intrusion detection, we attempt to open the backdoor in the exploitable vsftpd server.

Start ftp server:

```
mininet> h5 vsftpd &
```

Attempt the exploit:

```
mininet> h20 ftp h5
Name (10.0.0.5:vagrant): x:)
Password: any
$ nc 10.0.0.5 6200
```

Once the ftp client hangs the backdoor will be open on port 6200, access with:

Detection We assume that we know there is an FTP server on the network so we mirror all port 21 traffic. Alternatively, to be more precise mirroring can be done on the server itself. For information on what these look like, go to the IPFIX thresholds tab on the GUI or look in examples/threshold.yaml.

Attempting the exploit will the trigger the following snort rule:

```
alert tcp any any -> any 21 (msg:"VSFTPD Backdoor";
flow:established,to_server; content:"USER "; depth:5;
content:"|3a 29|"; distance:0; sid:2013188; rev:1;)
```

Note: the ftp client seems to be miss-calculating the checksum on USER and PASS packets (the packets we want to detect). By default snort does not consider packets with invalid checksums, so the following config is needed (within rules file):

The snort alert is then matched with the following snort threshold which blocks all communication from the host attempting the exploit, therefore blocking the backdoor. The $ symbol takes the sourceIPv4Address from the alert message (taken from the packet that triggered the alert).

```
alertmsg: 'VSFTPD Backdoor',
priority: 10,
treatment: block,
treatment_fields: {sourceIPv4Address: $}
```

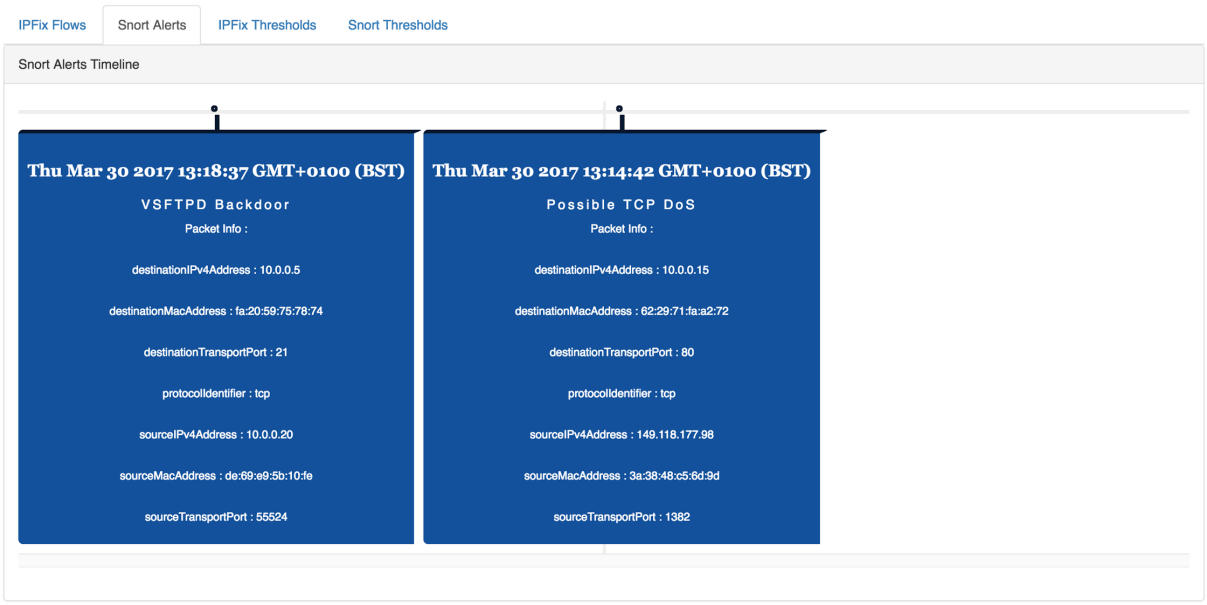
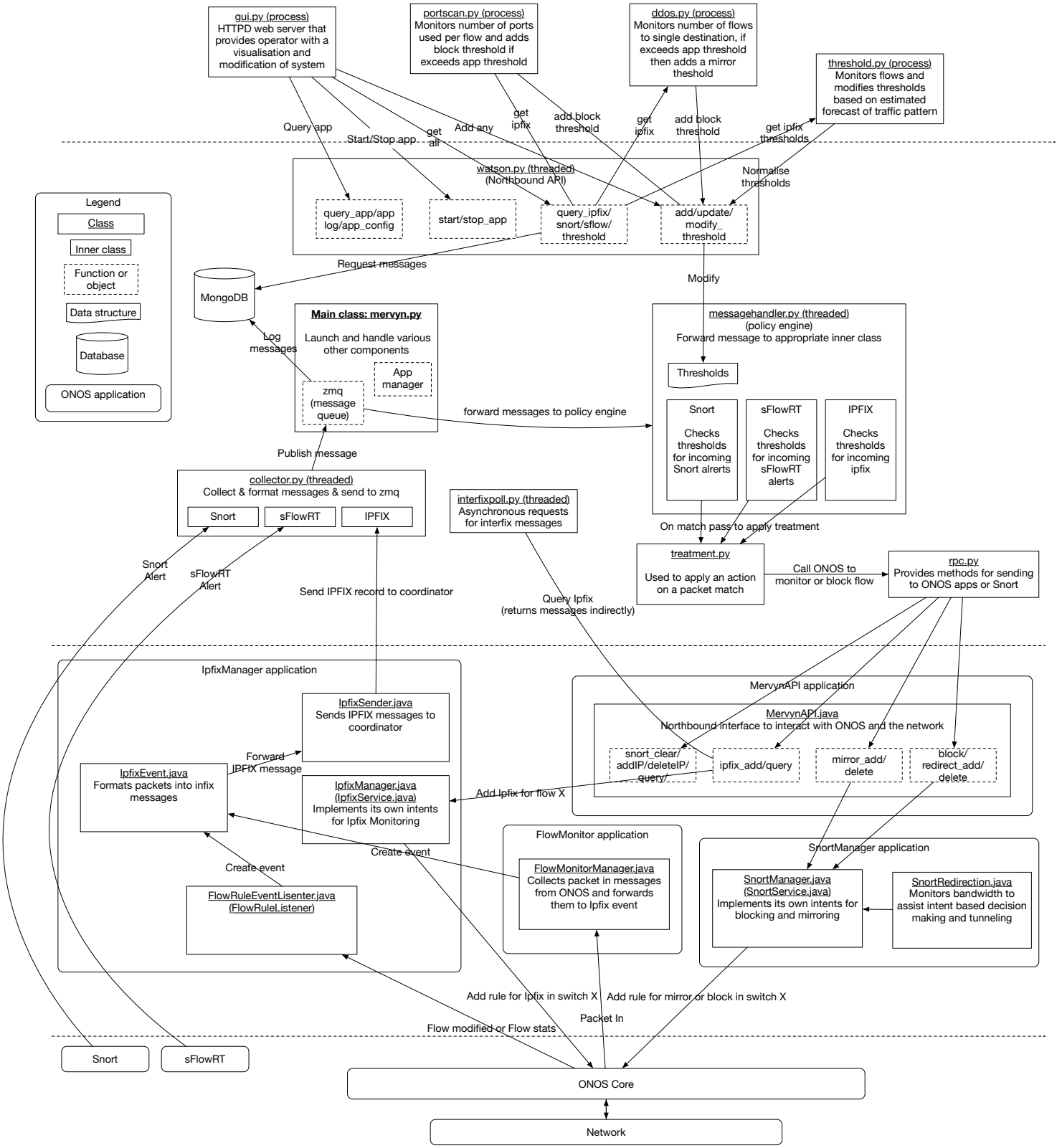
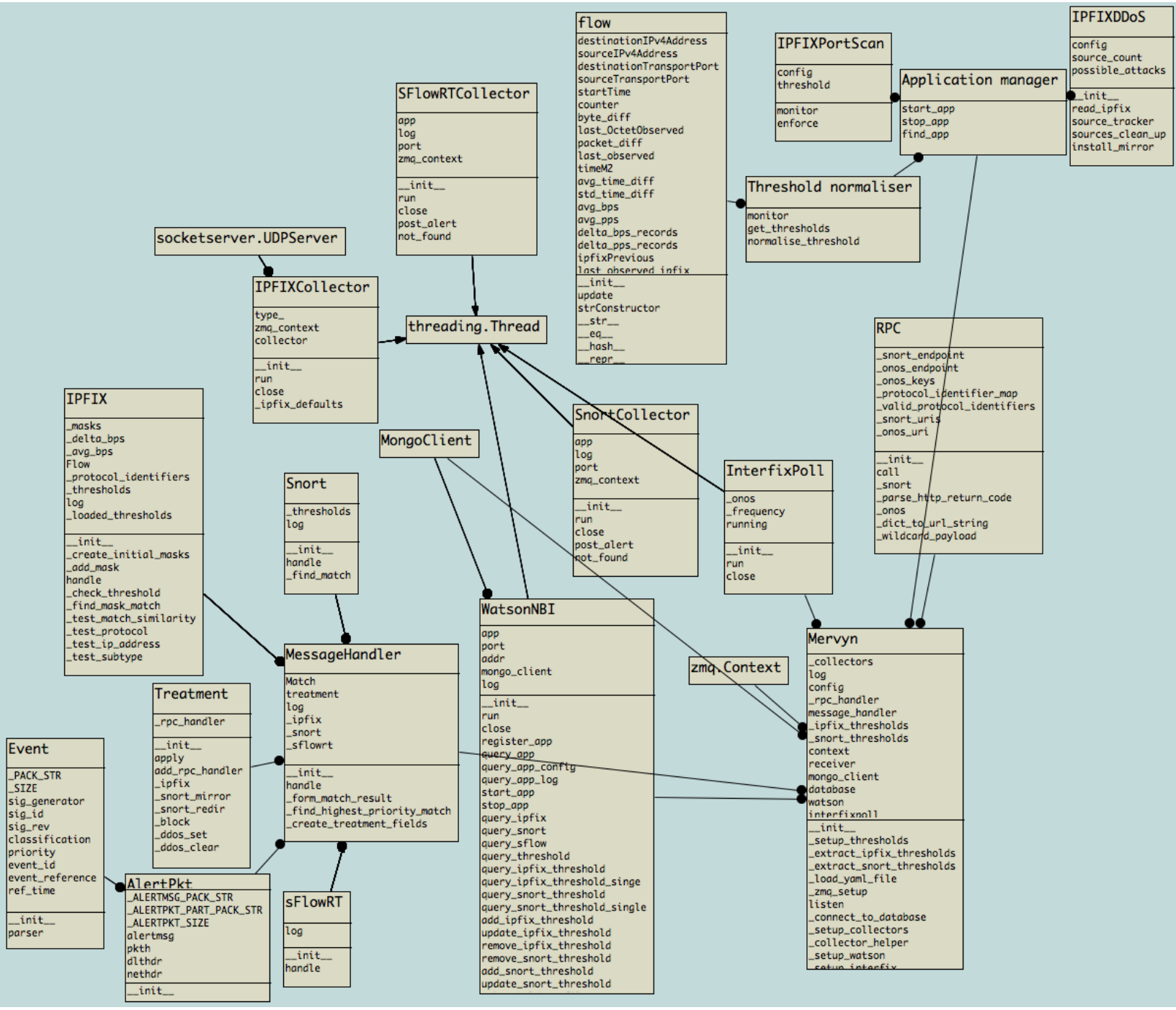


Figure 3: Screen capture of GUI displaying snort VSFTP alert

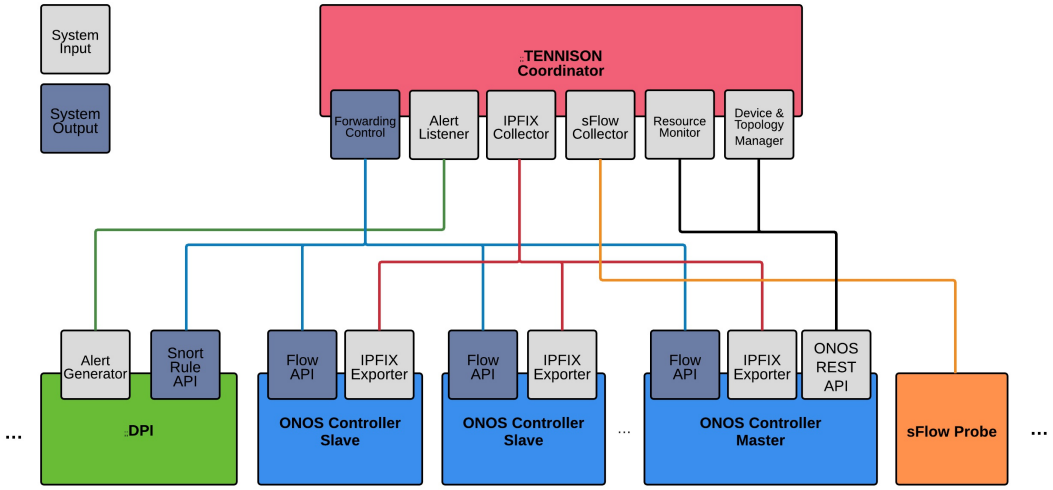
A TENNISON technical overview



B Class diagram of coordinator



C TENNISON flow API



TENNISON Overall Architecture (Modified with additional DPI input API)

ONOS Flow API

The coordinators access into ONOS for monitoring, remediation and snort management. Building on top of the Mervyn ONOS application (mervynapi)

Endpoint: http://onos:8181/mervyn

Monitoring

Path	Type	Description	Parameters
/ipfix/add/ `<saddr>` / `<sport>` / `<daddr>` / `<dport>` / `<protocol>`	GET	Adds an IPFIX monitoring intent	

			`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards not permitted**
/ipfix/query	GET	Triggers interfix messaes to be sent that match the specified flow.	None
/ipfix/query/flow/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Query a specific flow and send the necessary interfix message.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/redirect/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds redirection monitoring intent	5-tuple flow
/redirect/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET		

		Removes all redirection rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/mirror/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds mirror monitoring intent	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/mirror/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Removes mirror rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**

Remediation

Path	Type	Description	Parameters
/block/add/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Adds block remediation intent.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**
/block/delete/{saddr}/{sport}/{daddr}/{dport}/{protocol}	GET	Removes block rules that match the given flow from every switch.	`<saddr>` - IP source address. e.g. 192.168.1.1. `<sport>` - Transport Source Port. e.g. 47393. `<daddr>` - IP destination address. e.g. 192.168.1.2. `<dport>` - Transport Source Port. e.g. 80. `<protocol>` - Transport Protocol. e.g. {TCP, UDP, ICMP, ICMP6} **Wildcards permitted**

Snort Management/Discovery

Path	Type	Description	Parameters
snort/add/ `<ip>`	GET	Adds a node to be considered as a Snort instance. This node will then be forwarded traffic using addMirrorRule. Added in phase 2.	`<ip>` - IP Address of snort instance
snort/query	GET	Returns snort instances that have been added and discovered along with their connecting switch and port number	None
snort/del/ `<ip>`	GET	Removes the snort instance from above. Added in Phase 2.	`<ip>` - IP Address of snort instance
snort/clear	GET	Removes all snort instances from above	None

Examples

- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/ipfix/query'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/add/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/add/8.8.8.8/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/delete/8.8.8.8/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/block/delete/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/snort/add/10.0.0.1'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/add/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/remove/*/*/*/*/*'`
- `$curl -u karaf -X GET --header 'Accept: application/json' 'http://127.0.0.1:8181/mervyn/mirror/delete/*/*/*/*/*'`

Monitoring and Remediation Intents

The flow API allows monitoring to be inserted into the network in an intent based format. That is, when it is necessary to monitor a flow *somewhere* in the network it is requested that the flow is monitored by only providing the flow information. The placement of the monitoring mechanisms within the network is then choose by ONOS [application] though a series of data driven decisions. Following the'*what* not *how* (or *where* in our case)' paradigm, used by the forwarding intent based forwarding already provided by ONOS.

This approach aims to choose the optimal placement for monitoring whilst also reducing the total number of flow rules in the network when compared to the brute-force method of monitoring everything everywhere.

The same approach is also used for remediation. A description of how the placement is chosen for individual intents is outlined below.

IPFIX Monitoring intent

If using the L2 reactive forwarding application then an immediate suitable placement decision for an IPFIX monitoring rule cannot be made, due to the zero correlation between a L2 forwarding rule (source and destination MAC address) and IPFIX monitoring rule (source and destination, IP address and Transport port). Therefore an approach is used that results in an *eventual* single placement.

Initially the monitoring flow rule is installed on every device. Redundant rules are the removed over time in a garbage-collection-like process, until the flow is being monitored at a single device. The decision to remove rules is based on comparisons of their current activity (number of bytes and packets that have matched a flow); if a monitoring rule has less activity that the equivalent flow at a different device, it is considered redundant and therefore removed.

Redirect & Mirror intent

The number eligible devices for a redirection of mirroring rule are immediately reduced because of the requirement of having a Snort instance directly connected to the device. This set is then further reduced to devices that contain a flow rule that matches* that of the requested flow to be mirrored/redirect.

If this final set of eligible devices is not empty the redirect/mirror rule is install on a single device.

If no eligible devices were found for redirecting/mirroring, the flow is installed on all devices that have a matching* flow, in the hope that a snort instance will be later discovered at one of the devices.

As a last resort, if no devices contain a matching* flow the flow is install everywhere. This scenario should only really occur as a result of an error or manual request with no triggering traffic. With no matching traffic, the monitoring rules will eventually idle timeout.

Note: There could be some work here that uses topology and resource usage information to produce a cost value as an additional input to the decision making process.

Remediation intent

A block rule is installed onto **all** devices that currently contain a flow that matches* the requested flow to be blocked. Using this more inclusive method rather than remediating at a single device prevents a flow avoid the remediation itself by [re]routing.

* [Installed OpenFlow flow rule criteria is a superset of the requested flow]

D TENNISON northbound interface

TENNISON Northbound Interface

Endpoint: http://`<tennison>`:2401/

Messages and Alerts

Path	Type	Description	Parameters	Returns
/tennison/ipfix/query/`<app-id>`	GET	Retrieve IPFIX messages that have been received since the last query or registration.	`<app-id>` - Unique application identifier	JSON list of IPFIX message
/tennison/snort/query/`<app-id>`	GET	Retrieve Snort messages that have been received since the last query or registration.	`<app-id>` - Unique application identifier	JSON list of snort message (excluding packet data)

Thresholds

Path	Type	Description	Parameters	Returns
/tennison/thresholds/query	GET	Retrieve both IPFIX and snort thresholds	None	JSON list of thresholds
/tennison/thresholds/ipfix/query	GET	Retrieve IPFIX thresholds	None	JSON list of thresholds
/tennison/thresholds/snort/query	GET	Retrieve snort thresholds	None	JSON list of thresholds
/tennison/thresholds/ipfix/add/`<threshold-id>`	POST	Adds an IPFIX threshold	`<threshold-id>` - Unique threshold application identifier. Threshold - Content-Type: application/json. Required fields: subtype, treatment, fields, priority.	200 success 400 No content/Missing required fields

/tennison/thresholds/ipfix/update/ `<threshold-id>`	POST	Updates fields of an IPFIX threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json	200 success 400 No content 404 Threshold not found
/tennison/thresholds/ipfix/remove/ `<threshold-id>`	POST	Remove an IPFIX threshold	`<threshold-id>` - Unique threshold identifier.	200 success 404 Threshold not found
/tennison/thresholds/snort/add/ `<threshold-id>`	POST	Adds a snort threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json. Required fields: rule, treatment, fields, priority.	200 success 400 No content/Missing required fields
/tennison/thresholds/snort/update/ `<threshold-id>`	POST	Updates a snort threshold	`<threshold-id>` - Unique threshold identifier. Threshold - Content-Type: application/json	200 success 404 Threshold not found
/tennison/thresholds/snort/remove/ `<threshold-id>`	POST	Removes a snort threshold	`<threshold-id>` - Unique threshold identifier.	200 success 404 Threshold not found

Application Management

Path	Type	Description	Parameters	Returns
/tennison/app/register/ `<app-id>`	POST	Register application with tennison to be eligible for message from this point.	`<app-id>` - Unique application identifier	None
/tennison/app/query	GET	Get registered applications.	None	JSON list of application and the last query times
	GET	Get all configurations for application.		

/tennison/app/query/ `<app_id>` /config			`<app_id>` - Application ID	JSON list of application configurations.
/tennison/app/query/ `<app_id>` /log	GET	Get latest log for application.	`<app_id>` - Application ID	JSON containing log. e.g. {"log": "lorem ipsum"}
/tennison/app/start/ `<app_id>`	GET	Start application.	`<app_id>` - Application ID	200 success 400 Application already running 404 Application not installed
/tennison/app/stop/ `<app_id>`	GET	Stop application.	`<app_id>` - Application ID	200 success 400 Application already running 404 Application not installed

Examples

Register application

```
$ curl -X POST http://127.0.0.1:2401/tennison/app/register/ipfix-anom
```

Get registered applications

```
$ curl -X GET http://127.0.0.1:2401/tennison/app/query
```

Get IPFIX messages

```
$ curl -X GET http://127.0.0.1:2401/tennison/ipfix/query/ipfix-anom
```

Get all thresholds

```
$ curl -X GET http://127.0.0.1:2401/tennison/thresholds/query
```

Add new snort threshold

```
$ curl -X POST -H "Content-Type: application/json" -d '{"fields":{"sourceIPv4Address": "10.0.0.200"}, "rule":"alert tcp any any -> any any",
"priority":10, "treatment":"block"}' http://127.0.0.1:2401/tennison/thresholds/snort/add/new-snort-threshold
```

Add new IPFIX threshold

```
$ curl -X POST -H "Content-Type: application/json" -d '{"subtype": "ipfix", "treatment": "snort_mirror", "fields":{"sourceIPv4Address":
"10.0.0.255"}, "priority": 10}' http://127.0.0.1:2401/tennison/thresholds/ipfix/add/new-ipfix-threshold
```

Installing Northbound Applications

Although any application can register itself with the coordinator due to the openness of the REST interface, for an application to be considered as 'installed' and therefore managed by the coordinator there are a few requirements that need to be met.

- Location - each application has its own directory were all relevant files (executables, configs etc.) are located. Application directories should be in mervyn/apps with the directory name being the name of the application. E.g. mervyn/apps/ipfix-ddos-app/
- Main file - applications are launched via a python script `main.py` located in the application directory.
- Registering - when an application registers for monitoring messages (/tennison/app/register/ `<app-id>`) it should use the same ID as the application directory name.
- Config - application configurations should be stored in config.json
- Log file - any textual output from the application should go to output.log

Note: applications will not be launched with the working directory within the application directory, so avoid using relative file paths. An example of an applications directory structure is shown below.

```
mervyn
|    apps
|    |    ipfix-ddos-app
|    |    |    main.py
|    |    |    config.json
|    |    |    output.log
|    |    |    ipfixddos.py
|    |    another-application
```

E TENNISON collector API

IPFIX Collector

Uses sockets for the standard IPFIX transport method. UDP port 4739

Example pickled IPFIX data


```
{
         "vlanId" : 0,
         "ingressInterface" : 3,
         "subtype" : "ipfix",
         "flowStartMilliseconds":"2017-03-09T10:01:14.373000",
         "sourceTransportPort" : 0,
         "time":"2017-03-09T10:01:59.376000",
        "destinationMacAddress":"a6:49:e0:cc:4d:74",
         "flowEndMilliseconds":"2017-03-09T10:01:59.373000",
         "ipClassOfService" : 0,
         "packetDeltaCount" : 0,
         "egressInterface" : 0,
         "protocolIdentifier" : 1,
         "type" : "ipfix",
         "destinationTransportPort" : 0,
         "sourceMacAddress":"4e:ff:3c:1f:7e:91",
         "sourceIPv4Address" : "10.0.0.2",
         "exporterIPv4Address" : "127.0.0.1",
         "octetDeltaCount" : 0,
         "ethernetType" : 2048,
         exporterIPv6Address":"of:00:00:00:00:00:00:00:0b",
         "destinationIPv4Address" : "10.0.0.1"
}
```


Snort Alert Collector

REST API Port 8081

e.g. http://coord:8081

Path	Type	Description	Parameters

/snort/alert	POST	Adds snort alert onto the message queue	Content-Type: application/json

Example pickled snort alert


```
{
        "transhdr" : 0,
        "alertmsg" : "{'fields': {'sourceIPv4Address': '10.0.0.3'}, 'rule': 'alert icmp any any -> any any', 'treatment':
'snort_mirror', 'priority': 10}",
        "val" : 0,
        "event" : {
                "event_reference" : 131072,
                "ref_time" : {
                        "tv_sec" : 1747356,
                        "tv_usec" : 1477890048
                },
                "sig_rev" : 65536,
                "event_id" : 131072,
                "sig_generator" : 65536,
                "sig_id" : 10651137,
                "priority" : 0,
                "classification" : 0
        },
        "pkt" : {
        "sourceIPv4Address" : "10.0.0.2",
        "destinationIPv4Address" : "10.0.0.1",
        "protocolIdentifier" : "icmp",
        "sourceTransportPort" : 0,
        "destinationTransportPort" : 15
        },
        "pkth" : {
                "caplen" : 382730240,
                "len" : 0,
                "ts" : {
                        "tv_sec" : 447323224,
                        "tv_usec" : 0
                }
        },
        "data" : 973078528,
        "nethdr" : 0,
        "dlthdr" : 0,
        "type" : "snort",
        "time" : ISODate("2017-02-09T17:38:34.170Z")
```

```
}
```

sFlow Collector

REST API Port 8082

e.g. http://coord:8082

Path	Type	Description	Parameters
/sFlowRT/alert	POST	Adds sFlow-RT alert onto the message queue	Content-Type: application/json

Example sFlow-RT alert

```
{
     "agent":"10.80.80.188",
    "dataSource":"49",
    "metric":"ddos_blackhole_target",
    "threshold":20000,
    "value":20065.11977859513,
    "timestamp":1485861345308,
    "thresholdID":"ddos_blackhole_attack",
    "eventID":4,
    "flowKey":"10.80.80.51,external",
    "action":"ddos_set"
}
```

```
{
    "metric":"ddos_blackhole_target",
    "timestamp":1485863215066,
    "flowKey":"10.80.80.51,external",
    "action":"ddos_clear"
}
```

F Truffle API

Snort Rule API (Truffle)

REST API Port 8082

e.g. http://snort:8082

e.g. http://92.168.100.2:8082

Path	Type	Description	Parameters
/snort/rule/clear	POST	Clear all rules from snort	None
/snort/rule/delete	POST	Delete specific rule from snort	Content-Type: application/json { "rules" : ["rule 1", "rule 2] }
/snort/rule/add	POST	Add rule to snort	Content-Type: application/json { "rule" : ["rule 1", "rule 2] }