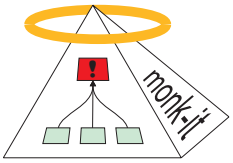


"monk-it"

Vermont Documentation

University of Erlangen
Computer Networks and Communication Systems
and Regional Computing Center

Authors

Tobias Limmer
Falko Dressler

23.12.2008

Contents

1. Overview

This document describes Versatile Monitoring Toolkit (Vermont) and the Vermont management infrastructure. The following sections give an overview of both systems. Afterwards, a detailed description of each system is presented.

1.1. Vermont

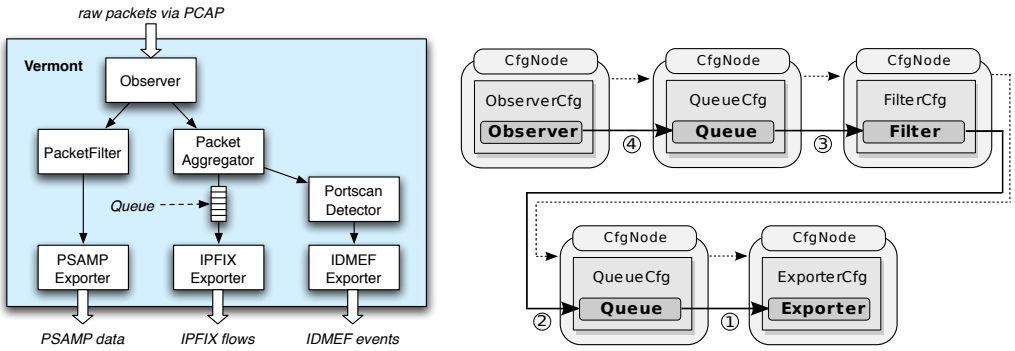
Vermont [LSMD06] is an open-source monitoring toolkit capable of processing Netflow.v9 and Internet Protocol Flow Information Export (IPFIX) conforming flow data. It has been developed in collaboration with the University of Tübingen[DC05]. The application runs on Linux and derivatives of BSD. It can receive and process raw packets via Packet Capturing (PCAP) (up to 1 GBit/s) as well as IPFIX/Netflow.v9 flow data. Supported data formats for export are IPFIX [Cla08, QBC+08], Packet Sampling (PSAMP) [Cla07, DCA+08], and Intrusion Detection Message Exchange Format (IDMEF) [DCF07]. The following modules are available:

- *Importers* capture raw data via PCAP, receive Netflow.v9 and IPFIX flow data via UDP and Stream Control Transmission Protocol (SCTP)
- *Samplers and filters* provide sampling algorithms and packet filter definitions
- *Exporters* export data using IPFIX, PSAMP, or IDMEF
- *Aggregators* aggregate incoming data according to customizable rules [DM06, DSMK08]; BiFlows [BT08] are also supported
- *Analyzers* detect anomalies in flows and output IDMEF events

Modules can be linked in almost any combination: only the input and output data type of linked modules need to be compatible. Modules may also have more than one succeeding and preceding module. Figure 1.1a shows an example for an arrangement of several modules. In this configuration, Vermont captures packets using PCAP, filters these packets and exports the selected PSAMP records. A second branch aggregates flows, which, in turn, are exported using IPFIX and analyzed in a portscan detector, respectively.

1.1.1. Reconfiguration

A special feature of Vermont is its support for dynamic reconfiguration of the module structure [LD09]. Linked modules in Vermont correspond to a directed acyclic graph and operate independently from each other.



(a) Sample module graph (b) Configuration and data paths

Figure 1.1.: Vermont module configuration

The idea is to support updates of the configuration file and to reconfigure Vermont accordingly at runtime. For this reconfiguration, Vermont computes the differences between the old and new configuration. Unique IDs are used to identify the modules. Vermont always tries to reuse existing modules in order to allow keeping state information and to speed up the reconfiguration process. If the configuration of an existing module has been changed, Vermont tries to reuse it and applies updates on-the-fly. If it is not possible to reuse a module, a new one is created. Examples are aggregator modules: for aggregation configurations, no on-the-fly reconfiguration is allowed because the used hash tables need to be rebuilt. Thus, all stored flows need to be exported and sent to the subsequent module in the module graph. This ensures as little flow data loss as possible. This process is repeated for each module until instances for all new modules are created. Modules are reconnected according to the new configuration and started in reverse topological order as depicted by the numbers in Figure 1.1b.

If modules do not have any asynchronous tasks to perform, they may be executed synchronously using a single thread. If, on the other hand, Vermont runs on a multicore machine, the software can be configured to use multiple threads, at most one per module. Asynchronous execution of modules causes lags in the processing time, so Vermont may use queues between modules to compensate this problem. The queues can be fully customized, but usually FIFO scheduling with a configurable size is used. The queues block if the maximum size is reached.

Figure 1.1b shows a configuration consisting of three modules that are connected by queues. Shown are the configuration paths (dashed lines) that link all the modules in the module graph and the data paths (thick lines) that depict the data flow between the modules.

The development of the reconfiguration process focused on minimizing the time during which data processing is stopped. It is technically not feasible to provide completely uninterrupted processing because the dependencies between the modules need to be considered. Especially, it is not possible to reconfigure the module graph without stopping the modules that need to be re-ordered in the graph. We minimized the module's outage by preparing new modules before the processing is stopped. Additionally, the shutdown

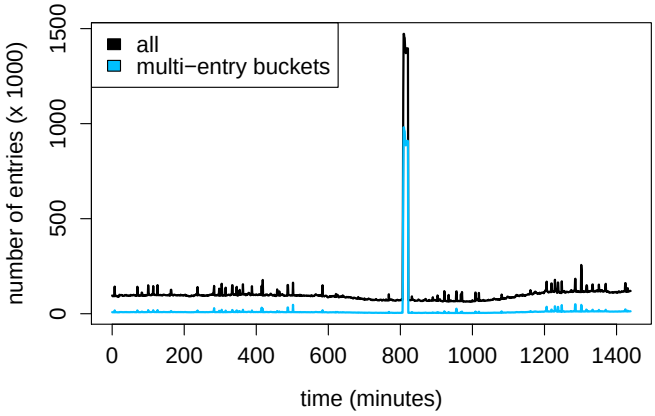


Figure 1.2.: Hash table size

of old modules is performed after the new configuration is completed and started.

We achieved downtimes smaller than 5 ms using this method. On a link transferring 1 GBit/s, this timeout could result in a data loss of about 650 KByte. Vermont is able to buffer this data during the reconfiguration process using the memory-mapped PCAP library.[1] For our tests, this buffer was set to 64 MByte.

1.1.2. Situation Awareness

Dynamic adaptation to current traffic data rates and corresponding load on flow meters does not only depend on seamless reconfiguration, but also on the ability to identify and, in the best case, anticipate bottlenecks in the monitoring hierarchy [LD09]. We implemented sensors inside Vermont to retrieve information about the current load of the system. Each module offers standard measurement values like CPU utilization and memory requirements. Additionally, module-specific data is monitored, e.g. the current packet rate or the queue size. This information is an essential requirement for algorithms that try to balance load among multiple flow aggregation nodes. Based on the data coming from the sensors, it is possible to move a task to a different system that still has unused capacities.

Figure 1.2 shows example statistics from the aggregator's hash table that were collected over one day: the black line shows the total number of entries inside the hash table, the blue line shows the number of entries that shared a single bucket with other entries inside the hash table. Multi-entry buckets considerably slow down the lookup of entries in a hash table, as they are implemented as linked lists. In our example, the hash table offered a total of 256 Kbuckets, but at the time of 800 min a DDoS attack occurred on the monitored link and the number of entries exceeded the hash table's capacity by far. This is a typical case for a DoS attack against the flow meter and should be evaded by monitoring the module load and adequate reconfiguration.

[1] `http://public.lanl.gov/cpw/`

1.2. Vermont Management

A common scenario in networks is the installation of monitoring systems at multiple points inside the network infrastructure. Here multiple Vermont instances need to be managed simultaneously. In this section, we will describe remote Vermont management tools that are able to supervise and edit the configuration of multiple Vermont instances, and to stop and to start them. Sensors inside Vermont can be read out and displayed in time-line graphs.

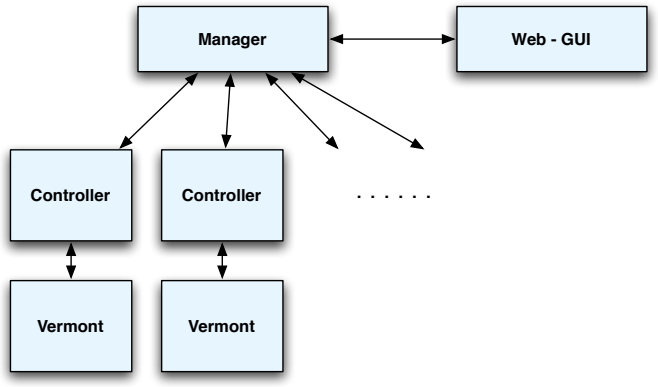


Figure 1.3.: Topology of the Vermont management system

The Vermont Management infrastructure consists of two central components: The web interface for controlling all available components and the manager which manages registration of all Vermont instances and controls them. The web interface only communicates with the management component and executes all tasks through it. Communication with single Vermont instances is performed by controllers which execute requests coming from the manager and relay them to each Vermont instance. Communication between web interface, manager and controller is performed using network-based Remote Procedure Call (RPC), so these components may be deployed on different hosts. The only restriction is the controller: it must be executed on the same host as Vermont, as communication between these components is performed by reading and writing files, as well as sending process signals. All commands initiated on the webinterface are forwarded to the manager which processes them directly, or forwards them to the addressed controller.

The Vermont management system is able to change configuration parameters of running Vermont instances on-the-fly based on observed sensor values. We created a closed loop that analyzes sensors values coming from specific Vermont instances and based on this data, we changed configuration parameters of running Vermont instances. Figure 1.4 displays the process.

Vermont's dynamic reconfiguration is realized using the so-called sensor actor concept. Sensors monitor input values from Vermont and are parametrized with a condition. If this condition is met, they will be triggered. Furthermore, the sensors are connec-

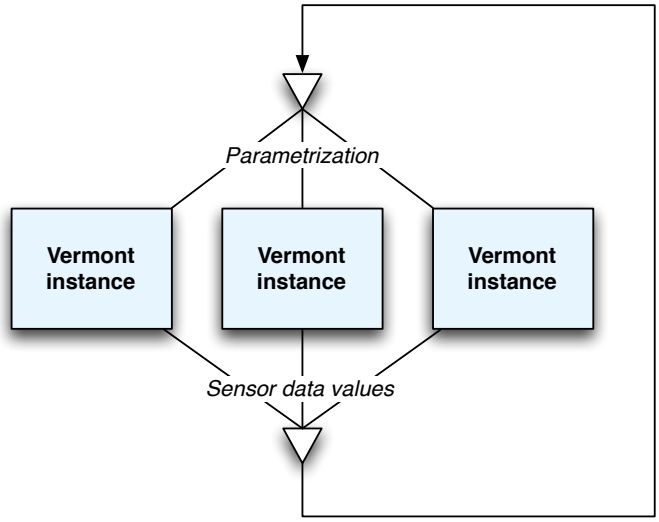


Figure 1.4.: Closed loop for Vermont reconfiguration

tion with one or more actors which execute an action when triggered. It is possible within Vermont to assign multiple actors to a sensor. All sensors and actors are directly parametrized inside the Extendible Markup Language (XML)-based configuration of the Vermont instance. Corresponding entries in these configuration files are ignored by Vermont and are only interpreted by the controller and manager components of the management infrastructure. Detailed description of the sensor-actor configuration can be found in section 3.2.4

1.2.1. Controller

Each instance of a Controller manages a single Vermont instance. For remote control over the network, a RPC interface is provided. The controller generates statistics from sensor values and stores them locally. When a request is received via the RPC interface, this data is accessed and transferred over the interface. The following functions are served via the interface:

- Retrieval of current state of Vermont
- Start and stop of Vermont
- Trigger of reconfiguration of Vermont
- Retrieval and change of provided configuration file
- Retrieval and change of the dynamic configuration file
- Retrieval of current sensor data values
- Retrieval of graphs showing the history of single sensor values

A second 'dynamic' configuration file was introduced for dynamic reconfiguration. If certain values are to be changed dynamically within the closed loop, only the dynamic configuration will be changed. The original configuration will be kept so that a configuration reset is always possible. The dynamic configuration file is always generated automatically and is initially a copy of the original configuration. It is always used by the running Vermont instance and uses the file name suffix `.dynconf`. The controller component has a seperate configuration file. The detailed structure is described in section 3.2.2.

1.2.2. Manager

Central management of multiple Vermont instances and their controllers is performed within the manager. The following tasks are carried out inside this component:

- Provides a directory of all configured vermont instances
- Dynamic reconfiguration that consists of the following tasks:
  - regular retrieval of all sensor values of the Vermont instances
  - check of all configured sensors
  - trigger of corresponding actors
  - execution of reconfiguration and transfer of updated configuration
  - trigger of configuration reload for concerned Vermont instances
- Provides an interface for the web interface

The manager usually only carries out regular management tasks and is the interface between web interface and controllers. This component does not require much performance and may be executed on the same host as the web interface.

1.2.3. Webinterface

The webinterface is available for easy configuration and control of used Vermont instances. Technically, the web interface is realized as `mod_python` module[2] for the webserver apache. All offered functions are relayed to the manager component and processed by it. In the following, we will describe the web interface and all pages within more thoroughly:

It is possible to view the status of all managed Vermont instances in an overview page. Figure 1.5 shows a screenshot of this page. This is the central management page and leads to all functions of the web interface. In detail, Vermont instances may be controlled (starting, stopping, reloading configuration), configurations may be edited, sensor data may be retrieved and displayed, and each Vermont instance's log file may be displayed. Dynamic configuration may also be switched on and off on this page. By switching it off, Vermont only uses static configuration and does not perform any dynamic reconfiguration tasks any more.

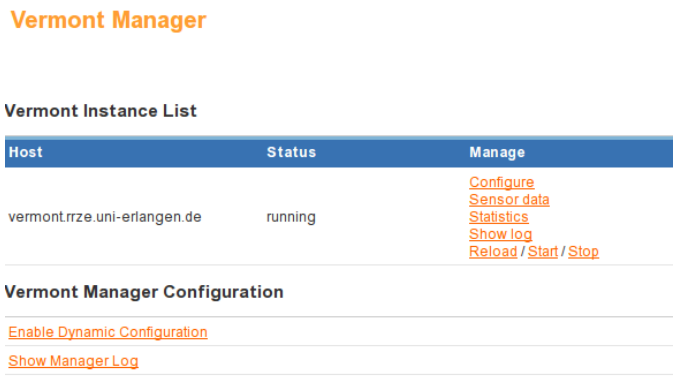


Figure 1.5.: Screenshot of overview page in the web interface

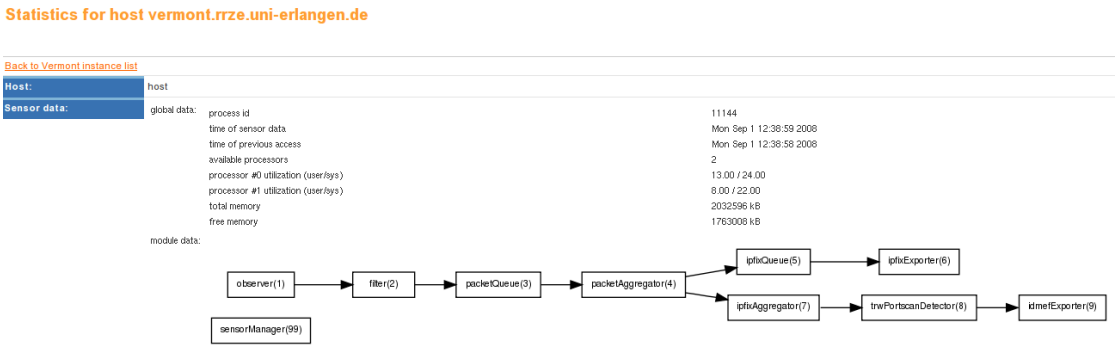


Figure 1.6.: Screenshot of page containing current sensor values

Current information can be retrieved by requesting generated statistics by the controller component, or by retrieval of current sensor data. Current sensor data may be displayed by selecting the link “Sensor Data”. Vermont produces a XML file containing current sensor values. This XML file is processed and converted to a Hypertext Markup Language (HTML) website with tables to give a good overview of the data, succeeded by the original XML data coming from Vermont for detailed inspection. At the top of the page, a graph is automatically generated showing the module graph that is currently used in Vermont (see also figure 1.6). To display the history of sensor values over time, the link “Statistics” leads to a page which displays graphs of sensor values of the corresponding Vermont instance. Figure 1.7 shows a screenshot of this page. These graphs are not generated over all available sensor data, but only from selected data that is specified in each controller component’s configuration.

When clicking on the “Configure” link in the overview page, the selected Vermont instance’s configuration is displayed and may be changed. Note that only valid XML data may be entered into this page, as this configuration is immediately parsed by the

[2] also see `http://www.modpython.org/`

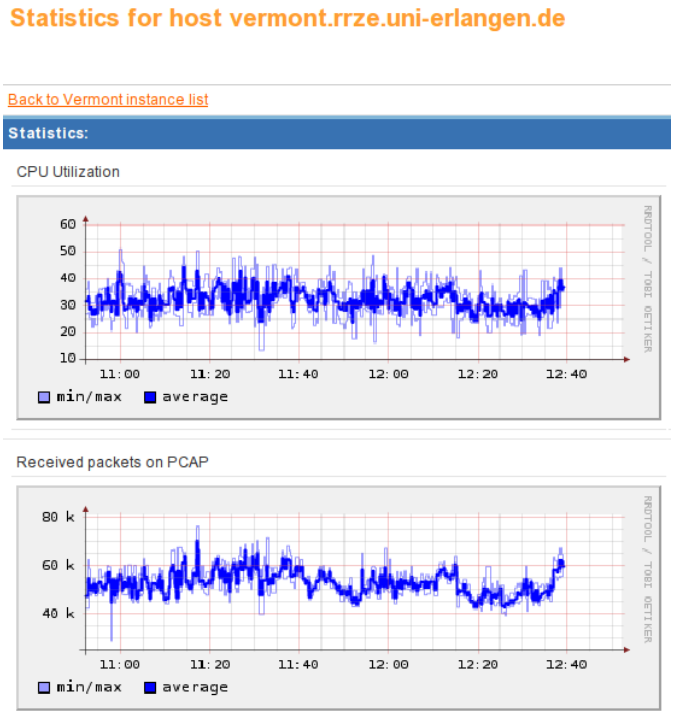


Figure 1.7.: Screenshot of page containing graphs of selected sensor values

manager component and is rejected if errors during the parsing process occur. This page additionally displays the currently used dynamic configuration, so it contains all parameter values changed by the sensor actor process.

2. Installation

This chapter describes the installation steps required for Vermont and its management infrastructure. We assume, that source code of both systems is already available in files `vermont.tar.bz2` and `vmanager.tar.bz2`.

2.1. Vermont

- The following tools and libraries are needed for Vermont (in their development version, i.e. including header files):
  - gcc for C++
  - cmake
  - libpcap
  - libpcre3
  - postgresql (>= 8.3, optional)
  - mysql (optional)
  - libboost
  - libxml2
  - libpq (optional)
  - libsctp (optional)
  - libpcap-mmap (optional)
- Extract tarball with `tar xjvf vermont.tar.bz2`.
- Go into directory `vermont` and execute `cmake .` to check for unresolved dependencies for compilation of Vermont.
- If Vermont needs to be compiled using the memory-mapped version of PCAP, please download and compile the library in directory `../libpcap-mmap` relative to the root of its Vermont's source code directory.
- Edit compilation parameters, most convenient way is by executing `ccmake .`
- Compile Vermont with `make` (on a multiple core system, `make -jX` triggers a parallel build with `X` parallel processes)
- Execute Vermont with example configuration and a little more information than usual (parameter `-d`): `./vermont -f configs/example.xml -d`

2.2. Vermont Management

- The following tools and libraries are needed for the Vermont management system:
  - python (>= 2.5)
  - apache2
  - libapache-mod-python
  - python-4suite-xml
  - python-pydot
  - graphviz
  - rrdtool
  - python-cheetah
  - python-dns
- Extract tarball with `tar xjvf vmanager.tar.bz2`
- Execute target creation script in source code root directory: `bash build_target.sh`
- Components manager (`target/manager`), controller (`target/controller`) and the webinterface (`target/webinterface`) are now ready-to-use in their described directories
- Edit configuration files for each of the components and distribute the controller to hosts
- Edit Apache configuration to include the Vermont webinterface in the web site. An example configuration is located in `target/webinterface/apache-config.sample`
- Run controllers and manager
- Access webinterface via `http://<servername>/<directory to webinterface>/start.py`

3. Structure and Configuration

3.1. Vermont

Vermont is heavily modularized. Every functionality is realized in corresponding modules, which have usually one input interface and one output interface. These two interfaces expect certain element types to be transferred. Within Vermont, the following element types are available:

- *Packet:* Represents a raw packet captured from the network interface or read from a dump file. Its class `Packet` provides pointers to detected headers like Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), etc. Usually these packets are not available in full length, but only the first N bytes (defined in configuration of module *Observer* and Vermont's global compile-time configuration.
- *IpfixRecord:* Represents a flow record. For each produced or incoming flow record, one element of this type is created. IPFIX templates are automatically extracted and are referenced within each IpfixRecord instance.
- *IdmefMessage:* Represents an IDMEF message in XML form. These elements are created by analyzer modules and contain a XML document describing the detected event.

Modules can be linked to a chain, so that the output of the preceding module is used as input for the successing module(s). Two modules may be only be connected, if they use the same output and input element type. All modules may have one or more output modules. In this case, all outgoing elements are not copied, but their references are copied to other modules. So if modules modify certain parts of the elements, other modules may be influenced by the modification. This aspect must be regarded when setting up the configuration and module structure. Modules are not allowed to receive input from multiple modules, as modules do not offer synchronization by default. For this case, queues can be prepended to the receiving module which perform synchronization tasks between all sending modules. If no queue is specified in front of a module that receives data from multiple modules, a queue with length of 1 is automatically inserted. To specify the connections between modules, each module ist identified by a unique ID number. For modules with successors, the IDs of the successing modules are listed in the module configuration as `<next>` tags.

Roughly, Vermont's available modules may be seperated into the following groups:

- *Packet Processing:* All modules processing raw packets.

  - Observer
  - PacketFilter
  - PacketQueue
  - PacketAggregator
  - PacketIDMEFReporter
  - PSAMPExporter
- *Flow Processing:* All modules processing flow records.
  - IpfixCollector
  - IpfixQueue
  - IpfixAggregator
  - IpfixExporter
  - IpfixPrinter
  - IpfixDbReader
  - IpfixDbWriter
  - IpfixDbWriterPg
  - RecordAnonymizer
- *IDMEF Processing:* All modules processing IDMEF messages.
  - IDMEFExporter
- *Flow Analyzer:* All modules analyzing flow records to produce events or statistics.
  - TRWPortscanDetector
  - Autofocus
- *Queues:* Queues that can connect multiple modules. In contrast to normal modules, they accept input from multiple modules and perform synchronization.
  - PacketQueue
  - IpfixQueue

3.1.1. IDMEFExporter

Exports incoming IDMEF messages to the external perl script idmefsender.pl which sends it over the network to a specified URL.

- *Input type:* IdmefMessage
- *Output type:* none

Example configuration:

```
<idmefExporter id="9">
    <sendurl>http://localhost</sendurl>
    <destdir>idmef_work</destdir>
</idmefExporter>
```

Parameters:

Parameter	Default value	Description
sendurl	none	Destination URL where IDMEF messages must sent to.
destdir	idmef_work	Directory, where IDMEF messages are temporary stored. There they are picked up by the external perl script idmefsender.pl in directory `/tools`.

3.1.2. IpfixAggregator

Aggregates incoming IPFIX flows according to specified parameters. Configuration is similar to module PacketAggregator.

- *Input type:* IpfixRecord
- *Output type:* IpfixRecord

Example configuration:

```
<ipfixAggregator id="6">
    <rule>
        <templateId>998</templateId>
        <biflowAggregation>1</biflowAggregation>
        <flowKey>
            <ieName>sourceIPv4Address</ieName>
        </flowKey>
        <flowKey>
            <ieName>destinationIPv4Address</ieName>
        </flowKey>
        <flowKey>
            <ieName>protocolIdentifier</ieName>
        </flowKey>
        <flowKey>
            <ieName>sourceTransportPort</ieName>
        </flowKey>
        <flowKey>
            <ieName>destinationTransportPort</ieName>
        </flowKey>
        <nonFlowKey>
            <ieName>flowStartMilliSeconds</ieName>
        </nonFlowKey>
        <nonFlowKey>
            <ieName>flowEndMilliSeconds</ieName>
        </nonFlowKey>
        <nonFlowKey>
            <ieName>octetDeltaCount</ieName>
        </nonFlowKey>
```

```
29          <nonFlowKey>
30              <ieName>packetDeltaCount</ieName>
31          </nonFlowKey>
32          <nonFlowKey>
33              <ieName>tcpControlBits</ieName>
34          </nonFlowKey>
35          <nonFlowKey>
36              <ieName>revflowStartMilliSeconds</ieName>
37          </nonFlowKey>
38          <nonFlowKey>
39              <ieName>revflowEndMilliSeconds</ieName>
40          </nonFlowKey>
41          <nonFlowKey>
42              <ieName>revoctetDeltaCount</ieName>
43          </nonFlowKey>
44          <nonFlowKey>
45              <ieName>revpacketDeltaCount</ieName>
46          </nonFlowKey>
47          <nonFlowKey>
48              <ieName>revtcpControlBits</ieName>
49          </nonFlowKey>
50      </rule>
51      <expiration>
52          <inactiveTimeout unit="sec">1</inactiveTimeout>
53          <activeTimeout unit="sec">1</activeTimeout>
54      </expiration>
55      <pollInterval unit="msec">1000</pollInterval>
56      <next>4</next>
57  </packetAggregator>
```

Parameters:

Parameter	Default value	Description
rule	none	Specifies a rule according to which is aggregated. More than one rule may be specified per aggregator.
biflowAggregation	0	Specifies if biflow aggregation is to be performed (0=no biflow, 1=biflow). Only valid in IpfixAggregator. To accomodate biflow information elements, Vermont-specific enterprise type ids were specified: revFlowStartMilliSeconds, revFlowEndMilliSeconds, revFlowStartSeconds, revFlowEndSeconds, revOctetDeltaCount, revPacketDeltaCount and revTcpControlBits.
templateId	none	Template ID (mandadory!).
flowKey		Flow key information element - flows are aggregated according to those keys.
nonFlowKey	none	Non-flow key information element - those IEs are aggregated.
ieName	none	name of the IE.
modifier	none	Optional field modifier for flow key IEs (”discard”, ”mask/X”).

match	0	Optional flow key filter for protocol identifier ("TCP", "UDP", "ICMP", or IANA number), IP addresses ("A.B.C.D/M"), port numbers (separated by ",", port range "A:B"), TCP control bits ("FIN", "SYN", "RST", "PSH", "ACK", "URG", separated by ",").
inactiveTimeout	0	Expiration timeout for idle/inactive flows.
activeTimeout	0	Periodic expiration timeout for long-lasting flows (typically larger than inactiveTimeout).
pollInterval	0	Length of interval when flows should be exported to next module.
hashtableBits	17	Length of hashtable used for aggregation in bits. The resulting hashtable will have a size of $2^{hashtableBits}$.

3.1.3. IpfixCollector

Receives IPFIX records from the network and imports them into Vermont. Protocols UDP and SCTP are supported at the moment.

- *Input type:* IdmefMessage
- *Output type:* none

Example configuration:

```
<ipfixCollector>
    <listener>
        <ipAddress>0.0.0.0</ipAddress>
        <transportProtocol>UDP</transportProtocol>
        <port>4739</port>
    </listener>
</ipfixCollector>
```

Parameters:

Parameter	**Default value**	**Description**
listener	none	Specifies a port where to listen for IPFIX flows.
ipAddress	none	IP address of interface on which collector receives IPFIX packets. If not given, collector receives at all interfaces.
transportProtocol	none	Must be set to 'UDP' or 'SCTP'.
port	4739	Port where Vermont listenes for incoming IPFIX flows.

3.1.4. IpfixExporter

Exports internal IPFIX records to the network using protocol UDP or SCTP.

- *Input type:* IpfixRecord
- *Output type:* none

Example configuration:

```
<ipfixExporter id="7">
    <templateRefreshInterval>10</templateRefreshInterval>
    <maxRecordRate>5000</maxRecordRate>
    <sctpDataLifetime unit="msec">10000</sctpDataLifetime>
    <sctpReconnectInterval unit="sec">30</sctpReconnectInterval>
    <collector>
        <ipAddressType>4</ipAddressType>
        <ipAddress>127.0.0.1</ipAddress>
        <transportProtocol>17</transportProtocol>
        <port>1500</port>
    </collector>
</ipfixExporter>
```

Parameters:

Parameter	Default value	Description
observationDomainId	0	Observation Domain ID of the exporter.
templateRefreshInteval	20 s	Interval for periodic sending of templates.
templateRefreshRate	10000	Interval for periodic sending of templates in records.
ipAddressType	4	Currently, only IPv4 is supported.
ipAddress	none	IP address of the collector the packets are sent to.
transportProtocol	none	Currently, only UDP (17) is supported.
port	4739	Port number of the collector.
maxRecordRate	5000	Maximum number of flow records per second sent to collector.
sctpDataLifetime	10 000 ms	Time how long SCTP considers a packet valid and tries to retransmit it.
sctpReconnectInterval	30 s	Time that Exporter waits to reestablish a lost connection.

3.1.5. IpfixPrinter

Prints incoming Ipfix flows to stdout for debugging purposes.

- *Input type:* IpfixRecord
- *Output type:* none

Example configuration:

```
<ipfixPrinter id="8">
    <lineOutput>1</lineOutput>
</ipfixPrinter>
```

Parameters:

Parameter	Default value	Description
lineOutput	0	Specifies if a special one-line-per-flow output should be used if value equals 1.

3.1.6. IpfixDbReader

Imports IPFIX flows from a MYSQL database table.

- *Input type:* none
- *Output type:* IpfixRecord

Example configuration:

```
<ipfixDbReader id="10">
    <host>127.0.0.1</host>
    <port>3306</port>
    <dbname>flows</dbname>
    <username>vermont</username>
    <password>v_password</password>
    <timeshift>true</timeshift>
    <next>12</next>
</ipfixDbReader>
```

Parameters:

Parameter	Default value	Description
host	none	Host of MySQL database.
port	3306	Port number of database.
dbname	none	Database name.
username	none	Username for database access.
password	none	Password for database access.
timeshift	false	Shift time stamps to current time.
fullspeed	false	If true, tables are read at full speed. Timeshifts are disabled. Otherwise, records are read from table approximately at the same speed as they were originally exported.
observationDomainId	0	Observation Domain Id assigned to the records.

3.1.7. IpfixDbWriter

Exports IPFIX flows to a database table in a MySQL database.

- *Input type:* IpfixRecord
- *Output type:* none

Example configuration:

```
<ipfixDbWriter id="10">
    <host>127.0.0.1</host>
    <port>3306</port>
    <dbname>flows</dbname>
    <username>vermont</username>
```

```
    <password>v_password</password>
    <bufferrecords>30</bufferrecords>
    <columns>
        <name>firstSwitched</name>
        <name>bytes</name>
    </columns>
</ipfixDbWriter>
```

Parameters:

Parameter	Default value	Description
host	none	Host of MySQL database.
port	3306	Port number of database.
dbname	none	Database name.
username	none	Username for database access.
password	none	Password for database access.
bufferrecords	30	Amount of flow records to buffer until they are written to the database.
observationDomainId	none	Observation Domain Id overriding the value to the records.
name	none	Column name (see IpfixDbCommon.hpp, currently one of "srcIP", "dstIP", "srcPort", "dstPort", "proto", "dstTos", "bytes", "pkts", "firstSwitched", "lastSwitched", "firstSwitchedMillis", "lastSwitchedMillis", "exporterID", "tcpControlBits", "revbytes", "revpkts", "revFirstSwitched", "revLastSwitched", "revFirstSwitchedMillis", "revLastSwitchedMillis", "revTcpControlBits", "maxPacketGap")

3.1.8. IpfixDbWriterPg

Exports IPFIX flows to a database table in a PostgreSQL database.

- *Input type:* IpfixRecord
- *Output type:* none

Example configuration:

```
<ipfixDbWriter id="10">
    <host>127.0.0.1</host>
    <port>3306</port>
    <dbname>flows</dbname>
    <username>vermont</username>
    <password>v_password</password>
    <bufferrecords>30</bufferrecords>
</ipfixDbWriter>
```

Parameters:

Parameter	Default	Description

	value	
host	none	Host of MySQL database.
port	3306	Port number of database.
dbname	none	Database name.
username	none	Username for database access.
password	none	Password for database access.
bufferrecords	30	Amount of flow records to buffer until they are written to the database. To achieve high performance, a value from 1000 to 10000 is recommended.

3.1.9. IpfixPayloadWriter

Writes IPFIX records including front payload into files. Only the first N (to be set in parameters) flows in chronological order are regarded. It is expected, that incoming flows are biflows and include front payload. For each biflow, three files are generated: .info contains header information of the flow, two .payload files contain front payload in both directions.

- *Input type:* IpfixRecord
- *Output type:* none

Example configuration:

```
<ipfixPayloadWriter id="5">
     <destPath>payload_work</destPath>
     <filenamePrefix>vfp</filenamePrefix>
     <connNumber>10</connNumber>
     <ignoreEmptyPayload>1</ignoreEmptyPayload>
</ipfixPayloadWriter>
```

Parameters:

Parameter	Default value	Description
destPath	none	Relative path where output files are stored.
filenamePrefix	none	Prefix for generated filenames.
connNumber	none	Amount of connections that are recorded. If this parameter is set to 0, no sorting will be performed and all incoming flows will be directly written to filesystem.
ignoreEmptyPayload	false	Set to true if all connections/biflows with empty payload shoud be ignored.
ignoreIncompleteTCP	false	Set to true if all TCP biflows without SYN flags in both directions should be ignored.
password	none	Password for database access.
bufferrecords	30	Amount of flow records to buffer until they are written to the database.

3.1.10. IpfixQueue

Caches IPFIX records in a queue until next module is ready to process them.

- *Input type:* IpfixRecord
- *Output type:* IpfixRecord

Example configuration:

```
<ipfixQueue id="3">
    <maxSize>10</maxSize>
    <next>6</next>
</ipfixQueue>
```

Parameters:

Parameter	Default value	Description
maxSize	1	Maximum number of items in queue. If queue is full, no new packets are accepted and preceding modules are paused.

3.1.11. Observer

Captures raw packets using the PCAP interface.

- *Input type:* none
- *Output type:* Packet

Example configuration:

```
<observer id="1">
    <interface>eth1</interface>
    <pcap_filter>ip</pcap_filter>
    <next>2</next>
</observer>
```

Parameters:

Parameter	Default value	Description
interface	none	Interface PCAP listens to. Do not use in combination with parameter filename.
captureLength	128	Sets the capture length of each packet. Packets bigger than that size are truncated. ATTENTION: if payload is analyzed in later modules, this parameter needs to be large enough!
filename	none	Must be specified if Vermont needs to read from file and contains its filename. Do not use in combination with parameter interface.

pcap_filter	none	Filter specification which is passed to PCAP (usually 'ip' to only capture IP packets).
replaceTimestamps	false	If true, PCAP packet timestamps are replaced with current time. This parameter only applies to PCAP file reading.
offlineSpeed	1.0	Only applies to PCAP file reading. Sets the speed multiplier for offline PCAP file reading. A negative value means read as fast as you can.
offlineAutoExit	true	Only applies to PCAP file reading. Sets if Vermont should be shut down automatically after reading all PCAP file data.

3.1.12. PacketFilter

Forwards packets which match specified filter configuration and drops non-matching packets.

- *Input type:* Packet
- *Output type:* Packet

Example configuration:

```
<filter id="2">
    <countBased>
        <interval>4</interval>
        <spacing>2</spacing>
    <countBased>
    <timeBased>
        <interval>100</interval>
        <spacing>50</spacing>
    </timeBased>
    <stringBased>
        <is>bla</is>
        <isnot>blub</isnot>
        <is type="HEX">0xFF024F</is>
    </stringBased>
    <regexBased>
        <matchPattern>is\s*not</matchPattern>
    </regexBased>
    <next>3</next>
</filter>
```

Parameters:

Parameter	Default value	Description
countBased	none	Creates a count-based filter. Spacing defines the number of packets accepted at the beginning of the period, interval specifies the length of the period
timeBased	none	Creates a time-based filter. Spacing defines the number of milliseconds during which time all incoming packets are accepted at the beginning of the period, interval specifies the length of the period in milliseconds.

interval	none	Belongs to either countBased or timeBased filter. Specifies length of period.
spacing	none	Belongs to either countBased or timeBased filter. Specifies amount of time or number of packets accepted.
stringBased	none	Creates a string-based filter which scans for specified strings inside the packet payload. If more than one search element is specified, only packets will be forwarded which match *all* specifications.
is	none	Accepts packets which contain included ASCII string. If attribute "type" is set to "HEX", the tag's content MUST include a hexstring which specifies the binary data to be searched for.
isnot	none	Drops packets which contain included ASCII string. If attribute "type" is set to "HEX", the tag's content MUST include a hexstring which specifies the binary data to be searched for.
regexBased	none	Creates a regex-based filter which scans for specified regexes inside the packet payload. If more than one regex is specified, only packets will be forwarded which match *all* specifications.
matchPattern	none	Specifies a regular expression used by the regex-based filter.
stateConnectionBased	none	Creates a filter which searches for TCP connections and matches all packets that contain the first N payload bytes. It uses a determinstic algorithm that will consume all the memory necessary to store all seen TCP connections.
connectionBased	none	Same as stateConnectionBased filter, but uses a different algorithm for TCP connection tracking. The algorithm is probabilistic and uses a fixed amount of memory to store the TCP connections.
timeout	3	Belongs to either stateConnectionBased or connectionBased filter. Specifies the time in seconds a seen TCP connection request is valid, before it will time out.
bytes	100	Belongs to either stateConnectionBased or connectionBased filter. Specifies how much Payload should be exported in bytes.
hashFunctions	3	Belongs to connectionBased filter. Specifies the number of hash functions that are used to index the bloom filters.
filterSize	1000	Belongs to connectionBased filter. Specifies the size of the bloom filters that are used by the connection based filter.
exportControlPackets	true	Controls wether TCP control packets (SYN/FIN/RST) are exported by stateConnectionBased and connectionBased filter.
anonFilter	none	Specifies a filter that performs anonymization on captured network packets. Contains one or more anonFields. This tag can have several subtags. The subtags are the same ones that can be used in the RecordAnonymizer module
payloadFilter	none	Payload is dropped, when this filter is specified.

3.1.13. PacketQueue

Caches packets in a queue until next module is ready to process them.

- *Input type:* Packet
- *Output type:* Packet

Example configuration:

```
<packetQueue id="3">
    <maxSize>10</maxSize>
    <next>6</next>
</packetQueue>
```

Parameters:

Parameter	Default value	Description
maxSize	0	Maximum number of items in queue. If queue is full, no new packets are accepted and preceding modules are paused.

3.1.14. PacketAggregator

Aggregates incoming raw packets to flows according to specified parameters.

- *Input type:* Packet
- *Output type:* IpfixRecord

Example configuration:

```
<packetAggregator id="6">
    <rule>
        <templateId>998</templateId>
        <flowKey>
            <ieName>sourceIPv4Address</ieName>
        </flowKey>
        <flowKey>
            <ieName>destinationIPv4Address</ieName>
        </flowKey>
        <flowKey>
            <ieName>protocolIdentifier</ieName>
        </flowKey>
        <flowKey>
            <ieName>sourceTransportPort</ieName>
        </flowKey>
        <flowKey>
            <ieName>destinationTransportPort</ieName>
        </flowKey>
        <nonFlowKey>
            <ieName>flowStartMilliSeconds</ieName>
```

```
21          </nonFlowKey>
22          <nonFlowKey>
23              <ieName>flowEndMilliSeconds</ieName>
24          </nonFlowKey>
25          <nonFlowKey>
26              <ieName>octetDeltaCount</ieName>
27          </nonFlowKey>
28          <nonFlowKey>
29              <ieName>packetDeltaCount</ieName>
30          </nonFlowKey>
31          <nonFlowKey>
32              <ieName>tcpControlBits</ieName>
33          </nonFlowKey>
34      </rule>
35      <expiration>
36          <inactiveTimeout unit="sec">1</inactiveTimeout>
37          <activeTimeout unit="sec">1</activeTimeout>
38      </expiration>
39      <pollInterval unit="msec">1000</pollInterval>
40      <next>4</next>
41  </packetAggregator>
```

Parameters:

Parameter	Default value	Description
templateId	none	Template ID (mandadory!).
flowKey		Flow key information element - flows are aggregated according to those keys.
nonFlowKey	none	Non-flow key information element - those IEs are aggregated.
ieName	none	name of the IE.
modifier	none	Optional field modifier for flow key IEs (”discard”, ”mask/X”).
match	0	Optional flow key filter for protocol identifier (”TCP”, ”UDP”, ”ICMP”, or IANA number), IP addresses (”A.B.C.D/M”), port numbers (separated by ”,”, port range ”A:B”), TCP control bits (”FIN”, ”SYN”, ”RST”, ”PSH”, ”ACK”, ”URG”, separated by ”,”).
inactiveTimeout	0	Expiration timeout for idle/inactive flows.
activeTimeout	0	Periodic expiration timeout for long-lasting flows (typically larger than inactiveTimeout).
pollInterval	0	Length of interval when flows should be exported to next module.
hashtableBits	17	Length of hashtable used for aggregation in bits. The resulting hashtable will have a size of $2^{hashtableBits}$.

3.1.15. PacketIDMEFReporter

For each incoming packet an IDMEF message is generated. An extract of the packet payload called snapshot may be included in the IDMEF message.

- *Input type:* Packet
- *Output type:* IdmefMessage

Example configuration:

```
<packetIDMEFReporter>
    <snapshotoffset>12</snapshotoffset>
    <snapshotlength>20</snapshotlength>
    <analyzerid>idmefreporter</analyzerid>
    <idmeftemplate>idmef/templates/idmefreporter_template.xml ↘
        →</idmeftemplate>
</packetIDMEFReporter>
```

Parameters:

Parameter	Default value	Description
snapshotoffset	0	Byte offset from start of packet payload.
snapshotlength	0	Byte length of snapshot. If it exceeds packet length, snapshot will be truncated.
analyzerid	none	Analyzer ID that will be included in IDMEF message.
idmeftemplate	none	Path to template file for IDMEF message.

3.1.16. PCAPExporter

Exports incoming packets into a file in PCAP format.

- *Input type:* Packet
- *Output type:* none

Example configuration:

```
<pcapExporter>
    <filename>output.pcap</filename>
</psampExporter>
```

Parameters:

Parameter	Default value	Description
filename	none	Name of the output pcap file.
linkType	EN10MB	Data link type of the output file. Names are DLT_ names form the pcap man page with the DLT_ removed (see 'man pcap')
snaplen	PCAP_MAX_ CAPTURE_LENGTH	Snaplen for the pcap file

3.1.17. PSAMPExporter

Exports incoming packets as PSAMP records over the network.

- *Input type:* Packet
- *Output type:* none

Example configuration:

```
<psampExporter id="1">
    <observationDomainId>123</observationDomainId>
    <ipfixPacketRestrictions>
        <maxPacketSize>200</maxPacketSize>
        <maxExportDelay unit="msec">500</maxExportDelay>
    </ipfixPacketRestrictions>
    <packetReporting>
        <templateId>888</templateId>
        <reportedIE>
            <ieName>sourceIPv4Address</ieName>
        </reportedIE>
        <reportedIE>
            <ieName>destinationIPv4Address</ieName>
        </reportedIE>
        <reportedIE>
            <ieName>ipPayloadPacketSection</ieName>
            <ieLength>65535</ieLength>
        </reportedIE>
    </packetReporting>
    <collector>
        <ipAddress>127.0.0.1</ipAddress>
        <transportProtocol>UDP</transportProtocol>
        <port>4739</port>
    </collector>
</psampExporter>
```

Parameters:

Parameter	Default value	Description
observationDomainId	0	Observation Domain ID of the exporter.
ipfixPacketRestrictions	none	Restrictions for IPFIX packets.
maxPacketSize	none	Maximum size of IPFIX packets.
maxExportDelay	none	Maximum delay until IPFIX packet is sent to destination.
packetReporting	none	Specifies elements to be exported for one template.
templateId	0	Specifies template ID.
reportedIE	none	Specifies one information element to be reported.
ieName	none	IPFIX type id of element to be exported.
ieLength	none	Optional specification of element length (usually only used by "ipPayloadPacketSection".
collector	none	Contains specification of one destination for PSAMP records.
idAddress	none	IP address of destination.
transportProtocol	none	Transport protocol to be used. Currently only "UDP" is supported.

port	4739	Port of destination.
templateRefreshRate	5000	Number of records, until template is resent.
templateRefreshInterval	30 s	Time, until template is resent.

3.1.18. RecordAnonymizer

This module is capable of anonymizing arbitary fields within IPFIX-Records using different anonymization methods.

- *Input type:* IpfixRecord
- *Output type:* IpfixRecord

Example configuration:

```
<anonRecord id="3">
    <anonField>
        <anonIE>
            <ieName>sourceIPv4Address</ieName>
        </anonIE>
        <anonMethod>CryptoPan</anonMethod>
        <anonParam>insert key here</anonParam>
    </anonField>
    <anonField>
        <anonIE>
            <ieName>destinationIPv4Address</ieName>
            <ieLength>4</ieLength>
        </anonIE>
    <anonMethod>CryptoPan</anonMethod>
        <anonParam>insert key here</anonParam>
    </anonField>
    <copyMode>false</copyMode>
    <next>6</next>
</anonRecord>
```

Parameters:

Parameter	Default value	Description
anonField	none	Specifies one field and an anonymization method for that field. Contains one anonIE, one anonMethod and an optional anonParam tag.
anonIE	none	Specifies the information element that needs to be anonymized. Belongs to anonField.
ieName	none	Specifies the name of the field that needs to be anonymized. Belongs to anonIE.
anonMethod	none	Specifies the anonymization method that is used to anonymize a given header field. Belongs to anonField. Possible values are: BytewiseHashHmacSha1, BytewiseHashSha1, ConstOverwrite, ContinuousChar, HashHmacSha1, HashSha1, Randomize, Shuffle, Whitenoise, CryptoPan

anonParam	none	Specifies an optional parameter to the anonymization method. Different methods need different params. ByteWiseHashHmacSha1, HashHmacSha1 need an variable sized key. ConstOverwrite needs one character as parameter. CryptoPan needs an 32 bytes long parameter (16 bytes key, 16 bytes pad). For CryptoPan and ConstOverwrite, keys can be specified as normal text, or as hexadecimal string starting with '0x'.
copyMode	false	If true, the Record Anonymizer creates a copy of the incoming record and leaves the original record unchanged. Copy mode should be turned on if the original records are processed by other moduls as well.

3.1.19. SensorManager

Module which controls all sensors ("'Messfühler"') inside Vermont. It does not have any in- or output types and must not be connected to any other module. It is recommended to set its ID to 99 to express its special role. If this module is specified in the configuration, available sensors are activated and polled regularly. It may only be specified once.

- *Input type:* none
- *Output type:* none

Example configuration:

```
<sensorManager id="99">
    <checkinterval>2</checkinterval>
    <outputfile>sensor_output.xml</outputfile>
</sensorManager>
```

Parameters:

Parameter	Default value	Description
checkinterval	2	Interval in seconds, when all sensors are polled and the output file is written to.
outputfile	`sensor_output.xml`	Path to file, where sensor data is stored.
append	0	Set to 1 if output file should be appended to, and not overwritten.

3.1.20. TRWPortscanDetector

Detects horizontal portscans in incoming IPFIX flows. Attention: IPFIX flows must be aggregated to biflows. To achieve best results, flows should contain the following IEs:

- sourceIPv4Address
- destinationIPv4Address
- sourceTransportPort

- destinationTransportPort
- protocolIdentifier
- flowStartMilliSeconds
- flowEndMilliSeconds
- revFlowStartMilliSeconds
- revFlowEndMilliSeconds
- octetDeltaCount
- revOctetDeltaCount
- packetDeltaCount
- revPacketDeltaCount
- tcpControlBits
- revTcpControlBits
- *Input type:* IpfixRecord
- *Output type:* IdmefMessage

Example configuration:

```
<trwPortscanDetector id="8">
    <analyzerid>trwportscandetector</analyzerid>
    <idmeftemplate>idmef/templates/trwportscan_template.xml ↘
        →</idmeftemplate>
    <hashbits>20</hashbits>
    <timeexpirepending>86400</timeexpirepending>
    <timeexpirescanner>1800</timeexpirescanner>
    <timeexpirebenign>1800</timeexpirebenign>
    <timecleanupinterval>10</timecleanupinterval>
    <next>9</next>
</trwPortscanDetector>
```

Parameters:

Parameter	Default value	Description
analyzerid	none	Analyzer ID which is inserted into the generated IDMEF message.
idmeftemplate	none	Path to IDMEF template which is used to generate the IDMEF message.
hashbits	20	Amount of bits used for hashtable to contain watched IP addresses.
timeexpirepending	86400	Seconds, until non-classified inactive IP addresses are purged from table.

timeexpirescanner	1800	Seconds, until as portscanner classified IP addresses are purged from table.
timeexpirebenign	1800	Seconds, until as benign classified IP addresses are purged from table.
timecleanupinterval	10	Interval length in seconds, when IP address table is scanned for entries to be purged.

3.2. Vermont Management

This section covers all configuration files used and interpreted by the Vermont management infrastructure.

3.2.1. Manager

The Vermont manager component is configured by a single XML file specified as command parameter at startup.

Listing 3.1: Sample configuration for manager

```
1  [Global]
2  Interval=5
3  Logfile=manager.log
4  AllowedWebIP=127.0.0.1
5  BindAddress=127.0.0.1
6  ListenPort=8001
7
8  [VermontInstances]
9  host_1=http://vermont.monitor.de:8000
10 host_2=http://vermont2.monitor.de:8000
```

You find an example configuration in listing 3.1. It contains the following parameters:

- `Interval`: Specifies the interval of retrieving sensor values of the Vermont instances in seconds. After the data retrieval, reconfiguration sensors are checked and, if needed, actors are triggered.
- `Logfile`: Specifies the file, where the manager log is to be written to.
- `AllowedWebIP`: Specifies the IP address, which may access the RPC interface. Only one IP address may be specified, as only the web interface needs to access this interface.
- `BindAddress`: Specifies the IP address of the networking interface, where manager should listen for RPC connections. Specify nothing, if manager should listen on all available interfaces. Be very careful, if all interfaces are used by the RPC interface, as manager does not perform access control or authentication!
- `ListenPort`: Specifies the port where manager should listen on. Usually port 8001 is used for manager.

- `host_X`: Configures all Vermont instance's controller using an Uniform Resource Locator (URL), where the controllers' interfaces are available. `X` needs to be incremented starting at 1.

3.2.2. Controller

Like the manager, the controller is configured by a single XML file specified as command parameter when executing it.

Listing 3.2: Example configuration for controller

```
1  [Global]
2  VermontDir=../vermont
3  ConfigFile=vermont.conf
4  ControllerLogFile=vcontroller.log
5  VermontLogFile=tmp.log
6  AllowedManagerIp=1.2.3.4
7  BindAddress=127.0.0.1
8  ListenPort=8000
9
10 [Stats]
11 Interval=5
12 Name_1=CPU Utilization
13 XPath_1=sum(/vermont/sensorData/processor[@id="0"]/util)
14 Name_2=Received packets on PCAP
15 XPath_2=sum(/vermont/sensorData/sensor[@name="observer"] ↘
       →/addInfo/pcap/received[@type="packets"])
16 Name_3=Dropped packets on PCAP
```

An example for the configuration file is displayed in listing 3.2. The following parameters may be specified:

- Sektion `Global`
  - `VermontDir`: Directory, in which the Vermont instance that needs to be managed is located (either absolute or relative to the current working directory).
  - `ConfigFile`: File name relative to path *VermontDir*, in which Vermont's configuration is saved.
  - `ControllerLogFile`: File name relative to the controller's working directory, in which the controller's log is saved.
  - `VermontLogFile`: File name relative to path *VermontDir*, in which all log data of the started Vermont instance is saved.
  - `AllowedManagerIp`: Specifies the IP address which may access the RPC interface. Only one IP address may be specified, as only the manager needs to access this interface. Ensure, that only one manager component accesses a controller, as two simultaneous dynamic configuration processes would influence each other unpredictably.

```
1 [Global]
2 VManager=http://localhost:8001
```

Figure 3.1.: Example configuration for web interface

  - `BindAddress`: Specifies the IP address of the networking interface, where controller should listen for RPC connections. Specify nothing, if controller should listen on all available interfaces. Be very careful, if all interfaces are used by the RPC interface, as controller does not perform access control or authentication!
  - `ListenPort`: Specifies the port where the controller should listen on. Usually port 8000 is used for the controller.

- Sektion `Stats`: Specifies sensor data in an enumerated list starting at 1. For these sensor data items, statistical data is saved to construct a time-line graph in the web interface. All specified terms "X" must be replaced by a number.
  - `Name_X`: Name of sensor data item
  - `XPath_X`: XPath String[1], which specifies the sensor data item within the XML file filled by Vermont with sensor data.
  - `Interval`: Interval in seconds, that specifies how often sensor data is retrieved for saving the statistics.

3.2.3. Webinterface

The Vermont webinterface is also configured by a single XML configuration file. It must be named `vermont_web.conf` and be located in the same directory as the script file for the web interface `start.py`. It only consists of a single parameter, as seen in listing 3.1:

- `VManager`: Specifies the URL, where the manager component's RPC interface can be found. Usually port 8001 is used for this interface.

3.2.4. Sensor-Actor System

The complete reconfiguration process is performed in component manager. In regular intervals (default: 5 seconds sensor data from all registered Vermont instances is retrieved, analyzed by sensors and eventually specified actors are triggered which modify configuration parameters of Vermont instances. Configuration of both sensors and actors is specified in each Vermont instance's configuration file and interpreted by the central manager component. Vermont ignores all parameters belonging to sensors and actors. In the following, we will describe possible configuration options for the reconfiguration framework.

[1] also see `http://www.w3.org/TR/xpath`

Sensors

Sensors read sensor data from a signle Vermont instance and acts on it. Source for sensor data is the XML file produced by Vermont's module called `SensorManager`. This XML file is generated regularly by Vermont, read by the controller and forwarded to the manager.

Listing 3.3: Example configuration for a sensor

```
<sensor id="1">
        <source>sum(/vermont/sensorData/processor/util)</source>
        <threshold>50</threshold>
        <activation>positive</activation>
</sensor>
```

Listing 3.3 show an example for a sensor's configuration. The follwoing parameters may be specified for each sensor:

- Attribute `id`: Each sensor is given an unique identification number. Attention: Dynamic reconfiguration acts globally on all Vermont instances. So this ID needs to be unique within the configurations of all Vermont instances.
- Element `source`: XPath-String[2], which specifies the data elements inside the XML document containing the sensor data coming from Vermont (File |sensor_output.xml—). This data is used as source for the corresponding sensor. A floating-point number is expected from Vermont manager as a result.
- Element `threshold`: Defines a limit, when all assigned actors are to be notified.
- Element `activation`: This value defines, if the sensor is activated when the the current value is above ("`positive`") or below ("`negative`") the limit, or never ("`disabled`").

In the displayed example, the sum of the utilization of all the system's processors is used as source. If this value is greater than 50, all connected actors are triggered.

During each check of the sensor values, the sensors trigger their assigned actors if the trigger conditions are satisfied. During a regular reconfiguration interval of 5 seconds, it is possible that actors are triggered in the same interval of 5 seconds.

Actors

Actors define actions that are executed after being triggered by a sensor. At the moment, the following actions are available:

[2]XPath is a string which defines a single or multiple elements within a XML document. It is similar to a path definition in a file system. In XPath, there are some additional functions specified to convert or aggregate multiple values inside the XML document. The XPath string specified in the example would select all sensor data values that are located in path |/vermont/sensorData/processor/util—. If multiple elements are found, their contents are summed up by the function |sum—. Look at `http://www.w3.org/TR/xpath` for a detailed description.

- change of parameters in configuration
- deactivation of Vermont modules
- reactivation of Vermont modules

1

Listing 3.4: Example configuration for an actor

```
<actor id="1">
        <action>modifyvalue</action>
        <code>v = int(v)*2</code>
        <trigger>always</trigger>
        <target>/ipfixConfig/filter[@id=2]/countBased/interval</target>
</actor>
```

Actors are configured in the same way as sensors directly in the XML configuration of Vermont. Listing 3.4 shows a configuration example. Each actor accepts the following parameters:

- Attribute `id`: This ID connects an actor with a sensor. If an actor uses the same ID as a sensor, the actor will be triggered by the sensor. It is possible to connect multiple actors by using the same ID.
- Element `action`: Defines an action that is executed after the actor was triggered by a sensor. The follwing actions are implemented:
  - `modifyvalue`: Changes a parameter in the Vermont configuration, specified in element `target`. How the value is changed is documented in element `code`.
  - `pausemodule`: Pauses the module specified in element `target` by removing all connections to preceding modules. Then the module is still activated inside Vermont, but does not receive any input data any more. If it is reactivated again, the module's state will be kept this way.
  - `resumemodule`: Reactivates the module specified in element `target`. All connections that were removed by action `pausemodule` are reinserted into the configuration, so that the module starts receiving input data again.
- Element `code`: This element parametrizes the method how to change the parameter value for action `modifyvalue`. This element expects Python code as input. Variable `v` contains the configuration parameter's current value. This value can be modified arbitrarily. The value of variable `v` will after execution of the code applied to Vermont's configuration.
- Element `trigger`: Specifies, how often the actor reacts on trigger events from sensors. The following values can be used:
  - `always`: When the actor is triggered by the assigned sensor, the configuration action is executed immediately.

  - `delayed`: If the actor is triggered over an interval of `delay`, the action will be executed once.
  - `once`: The actor's action will be executed, when the assigned sensor did not trigger in the preceding interval, but triggered in the current one.
  - `once delayed`: Is a combination of `delayed` and `once`: The actor will be executed when the assigned sensor was active and triggered the actor over an interval of `delay`. After executing the action, the actor is not activated any more, except the sensor was inactive for a duration of `delay`. From this moment on, the actor is activated again and may execute its action like previously described.

- Element `delay`: Defines a time interval in seconds that is used by the actions `delayed` and `once delayed`.

- Element `target`: When using the action `modifyvalue`, this element specifies a parameter inside Vermont's XML configuration file using XPath. The Path `/ipfixConfig/filter[@id=2]/countBased/interval`[3] specifies in this example, that the interval of a sampler is target for modification. When using the actions `pausemodule` and `resumemodule`, this element points to a configuration element specifying a Vermont module, as e.g. the XML element `<observer>`.

The displayed example configuration for an actor changes the parameter value of a filter module each time a sensor triggers it. The configured interval will be doubled each time.

If actors change Vermont's configuration, the configuration will be automatically transmitted to the running Vermont instance by updated the configuration file and a reconfiguration of Vermont is triggered by sending a process signal. These tasks are executed by the controller component.

[3] In this XPath string, `filter[@id=2]` specifies the XML element `filter` containing the value 2 in attribute `id`.

A. Appendix

A.1. Example configuration of a closed loop within a Vermont instance

In this section, we present a complete example configuration for Vermont including multiple sensors and actors. Two sensors use the processor utilization as input source: The sensor with ID 1 is activated, when the processor utilization increases above 50% and sensor with ID 2 is activated, if the processor utilization falls below 30%. The first two actors change the filter's parameters and duplicate / half the sampling rate each time they are triggered. The succeeding actors pause and reactivate the module `ipfixAggregator`.

```
<ipfixConfig>
    <sensors>
        <sensor id="1">
            <source>sum(/vermont/sensorData/processor/util)</source>
            <threshold>50</threshold>
            <activation>positive</activation>
        </sensor>
        <sensor id="2">
            <source>sum(/vermont/sensorData/processor/util)</source>
            <threshold>30</threshold>
            <activation>negative</activation>
        </sensor>
    </sensors>
    <actors>
        <actor id="1">
            <action>modifyvalue</action>
            <code>v = int(v)*2</code>
            <trigger>always</trigger>
            <target>/ipfixConfig/filter[@id=2] ↘
                →/countBased/interval</target>
        </actor>
        <actor id="2">
            <action>modifyvalue</action>
            <code>if int(v)>2: v = int(v)/2</code>
            <trigger>always</trigger>
            <target>/ipfixConfig/filter[@id=2] ↘
                →/countBased/interval</target>
        </actor>
        <actor id="1">
            <action>pausemodule</action>
            <trigger>once</trigger>
            <target>/ipfixConfig/ipfixAggregator[@id=7]</target>
```

```
31          </actor>
32          <actor id="2">
33              <action>resumemodule</action>
34              <trigger>once</trigger>
35              <target>/ipfixConfig/ipfixAggregator[@id=7]</target>
36          </actor>
37      </actors>
38
39      <sensorManager id="99">
40          <checkinterval>1</checkinterval>
41      </sensorManager>
42
43      <observer id="1">
44          <interface>eth1</interface>
45          <pcap_filter>ip</pcap_filter>
46          <next>2</next>
47      </observer>
48
49      <filter id="2">
50          <countBased>
51              <interval>2</interval>
52              <spacing>2</spacing>
53          </countBased>
54          <next>3</next>
55      </filter>
56
57      <packetQueue id="3">
58          <maxSize>1000</maxSize>
59          <next>4</next>
60      </packetQueue>
61
62      <packetAggregator id="4">
63          <rule>
64              <templateId>998</templateId>
65              <flowKey>
66                  <ieName>sourceIPv4Address</ieName>
67              </flowKey>
68              <flowKey>
69                  <ieName>destinationIPv4Address</ieName>
70              </flowKey>
71              <flowKey>
72                  <ieName>protocolIdentifier</ieName>
73              </flowKey>
74              <flowKey>
75                  <ieName>sourceTransportPort</ieName>
76              </flowKey>
77              <flowKey>
78                  <ieName>destinationTransportPort</ieName>
79              </flowKey>
80              <nonFlowKey>
81                  <ieName>flowStartMilliSeconds</ieName>
82              </nonFlowKey>
83              <nonFlowKey>
84                  <ieName>flowEndMilliSeconds</ieName>
```

```
85              </nonFlowKey>
86              <nonFlowKey>
87                  <ieName>octetDeltaCount</ieName>
88              </nonFlowKey>
89              <nonFlowKey>
90                  <ieName>packetDeltaCount</ieName>
91              </nonFlowKey>
92              <nonFlowKey>
93                  <ieName>tcpControlBits</ieName>
94              </nonFlowKey>
95          </rule>
96          <expiration>
97              <inactiveTimeout unit="sec">10</inactiveTimeout>
98              <activeTimeout unit="sec">60</activeTimeout>
99          </expiration>
100         <pollInterval unit="msec">10000</pollInterval>
101         <next>5</next>
102         <next>7</next>
103     </packetAggregator>
104
105     <ipfixQueue id="5">
106         <maxSize>100000</maxSize>
107         <next>6</next>
108     </ipfixQueue>
109
110     <ipfixExporter id="6">
111         <collector>
112             <ipAddressType>4</ipAddressType>
113             <ipAddress>10.1.1.1</ipAddress>
114             <transportProtocol>17</transportProtocol>
115             <port>1500</port>
116         </collector>
117         <maxRecordRate>10000</maxRecordRate>
118     </ipfixExporter>
119
120     <ipfixAggregator id="7">
121         <rule>
122             <templateId>999</templateId>
123             <biflowAggregation>1</biflowAggregation>
124             <flowKey>
125                 <ieName>sourceIPv4Address</ieName>
126             </flowKey>
127             <flowKey>
128                 <ieName>destinationIPv4Address</ieName>
129             </flowKey>
130             <flowKey>
131                 <ieName>protocolIdentifier</ieName>
132             </flowKey>
133             <flowKey>
134                 <ieName>sourceTransportPort</ieName>
135             </flowKey>
136             <flowKey>
137                 <ieName>destinationTransportPort</ieName>
138             </flowKey>
```

```
139             <nonFlowKey>
140                 <ieName>flowStartMilliSeconds</ieName>
141             </nonFlowKey>
142             <nonFlowKey>
143                 <ieName>flowEndMilliSeconds</ieName>
144             </nonFlowKey>
145             <nonFlowKey>
146                 <ieName>octetDeltaCount</ieName>
147             </nonFlowKey>
148             <nonFlowKey>
149                 <ieName>packetDeltaCount</ieName>
150             </nonFlowKey>
151             <nonFlowKey>
152                 <ieName>tcpControlBits</ieName>
153             </nonFlowKey>
154             <nonFlowKey>
155                 <ieName>revflowStartMilliSeconds</ieName>
156             </nonFlowKey>
157             <nonFlowKey>
158                 <ieName>revflowEndMilliSeconds</ieName>
159             </nonFlowKey>
160             <nonFlowKey>
161                 <ieName>revoctetDeltaCount</ieName>
162             </nonFlowKey>
163             <nonFlowKey>
164                 <ieName>revpacketDeltaCount</ieName>
165             </nonFlowKey>
166             <nonFlowKey>
167                 <ieName>revtcpControlBits</ieName>
168             </nonFlowKey>
169         </rule>
170         <expiration>
171             <inactiveTimeout unit="sec">60</inactiveTimeout>
172             <activeTimeout unit="sec">120</activeTimeout>
173         </expiration>
174         <pollInterval unit="msec">10000</pollInterval>
175         <next>8</next>
176     </ipfixAggregator>
177 
178     <trwPortscanDetector id="8">
179         <analyzerid>trwportscandetector</analyzerid>
180         <idmeftemplate>idmef/templates/trwportscan_template.xml ↘
                →</idmeftemplate>
181         <next>9</next>
182     </trwPortscanDetector>
183 
184     <idmefExporter id="9">
185         <sendurl>http://localhost</sendurl>
186     </idmefExporter>
187 </ipfixConfig>
```

Bibliography

[BT08] Elisa Boschi and B. Trammell. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103, IETF, January 2008.

[Cla07] Benoit Claise. Packet Sampling (PSAMP) Protocol Specifications. Internet-Draft (work in progress) draft-ietf-psamp-protocol-09.txt, IETF, December 2007.

[Cla08] Benoit Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, IETF, January 2008.

[DC05] Falko Dressler and Georg Carle. HISTORY - High Speed Network Monitoring and Analysis. In *24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005), Poster Session*, Miami, FL, March 2005. IEEE.

[DCA+08] Thomas Dietz, Benoit Claise, Paul Aitken, Falko Dressler, and Georg Carle. Information Model for Packet Sampling Exports. Internet-Draft (work in progress) draft-ietf-psamp-info-11.txt, IETF, October 2008.

[DCF07] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). Technical Report RFC 4765, IETF, March 2007.

[DM06] Falko Dressler and Gerhard Münz. Flexible Flow Aggregation for Adaptive Network Monitoring. In *31st IEEE Conference on Local Computer Networks (LCN): 1st IEEE LCN Workshop on Network Measurements (WNM 2006)*, pages 702–709, Tampa, FL, November 2006. IEEE.

[DSMK08] Falko Dressler, Christoph Sommer, Gerhard Münz, and Atsushi Kobayashi. IPFIX Flow Aggregation. Internet-Draft (work in progress) draft-dressler-ipfix-aggregation-05.txt, IETF, July 2008.

[LD09] Tobias Limmer and Falko Dressler. Seamless Dynamic Reconfiguration of Flow Meters: Requirements and Solutions. In *16. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2009)*, Kassel, Germany, March 2009. Springer. to appear.

[LSMD06] Ronny T. Lampert, Christoph Sommer, Gerhard Münz, and Falko Dressler. Vermont - A Versatile Monitoring Toolkit Using IPFIX/PSAMP. In *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*, pages 62–65, Tübingen, Germany, September 2006. IEEE.

[QBC+08] Jürgen Quittek, Stewart Bryant, Benoit Claise, Paul Aitken, and Jeff Meyer. Information Model for IP Flow Information Export. RFC 5102, IETF, January 2008.