Project Number :	IST-2002-002154
Project Title :	Distributed Adaptive Security by Programmable Firewall

DIADEM Firewall

D5 – Architecture Specifications

Deliverable Type:	Document
Dissemination:	Public
Contractual date:	January 2005

Editor:	Yannick Carlinet (FT)
File Name:	Diadem Firewall - D5 - Architecture Specifications.doc
Contributors:	See list of authors
Version:	Final
Version Date:	January 24, 2005
Deliverable Status:	Final

The DIADEM Firewall consists of:

	Partner	Short name	Country
1	France Telecom	FT	France
2	University of Tübingen	TU	Germany
3	IBM Research GmbH Zurich Research Laboratory	IBM ZRL	Switzerland
4	Imperial College London	Imperial	United Kingdom
5	Jozef Stefan Institute	JSI	Slovenia
6	Groupe des Ecoles des Télécommunications	GET	France
7	Polish Telecom	TP	Poland

Project Management:
Yannick Carlinet (FT)
Phone +33 2.96.05.03.25
Fax: +33 2 96 05 37 84
E-mail yannick.carlinet@francetelecom.com
France Telecom CORE/M2I
2 ave. Pierre Marzin,
22307 Lannion, France

List of authors:
Patricia Sagmeister, IBM ZRL
Roland Wehage, IBM ZRL
Gero Dittmann, IBM ZRL
Jan van Lunteren, IBM ZRL
Olivier Paul, GET
Sherif Yusuf, Imperial
Morris Sloman, Imperial
Vrizlynn Thing, Imperial
Gerhard Münz, TU
Falko Dressler, TU
Tomasz Kołoszczyk, TP
Michał Kowalczyk, TP
Dušan Gabrijelčič, JSI
Yannick Carlinet, FT

Executive summary
This document presents the architecture of the Diadem Firewall system as a whole. There are five high-level components: Devices, Monitoring Element, Firewall Element, Violation Detection, and System Manager. Devices are network equipments of the data plane which can be controlled by a component of the Diadem Firewall architecture. Monitoring Elements configure the monitoring functions of the Devices (e.g. Netflow for a Cisco router device) and collect the monitoring data issued by these devices. They aggregate the collected data and send it to the Violation Detection for analysis. Monitoring Elements are themselves configured by the Violation Detection (VD). The VD contains the attack detection logic. When an attack is detected, the VD sends a report to the System Manager. The System Manager analyses these reports and issues response policies to the Firewall Elements. The latter then interpret the response policies and convert them into configuration rules that are enforced on the appropriate Devices, in order to mitigate the attack.

Acronyms

ACL	Access Control List
ALG	Application Level gateway
API	Application Programming Interface
BaRT	Balanced Routing Table Search
CE	Classifier Engine
CLI	Command Line Interface
DB	Database
DDoS	Distributed Denial of Service
DoS	Denial of Service
HTTP	Hyper Text Transfer Protocol
IDS	Intrusion Detection System
IDMEF	Intrusion Detection Message Exchange Format
IPFIX	Internet Protocol Flow Information Export
FD	Firewall Device
FPGA	Field Programmable Gate-Array
FE	Firewall Element
MD	Monitoring Device
ME	Monitoring Element
NAT	Network Address Translation
P^2C	Parallel Packet Classification
PCI	Peripheral Component Interconnect
PM	Policy Manager
PMA	Policy Management Agent
SM	System Manager
SRAM	Static Random Access Memory
TCAM	Ternary Content Address Memory
VD	Violation Detection

Table of Contents

1 Introduction

The aim of this document is to present the architecture of the Diadem Firewall system as a whole. The architecture was designed with four high-level requirements in mind:

- DDoS (Distributed Denial of Service) can be detected and mitigated efficiently
- The system is flexible enough to be adapted to new upcoming attacks
- The system protects the broadband customers as well as the ISP and operator networks
- The architecture can be easily integrated in an existing operator infrastructure

Section 2 gives an overview of the architecture and presents the high-level components. The next five sections detail the five high-level components and show how they interact with each other.

2 Overview of the architecture

In the following, we separate the different components of the architecture according to three levels: data level, element level, administrative domain level. The data level contains all the equipments that deal with the actual traffic of the customers of the network operator. The element level is an abstraction layer for the data level equipments. The purpose of the components in the element level is to provide a unified interface to control the equipments in the data level, independently of their specific (sometimes proprietary) control interface. For instance in the case of the Diadem Firewall architecture, one of the roles of the element level components is to translate policies into configuration rules that comply with each underlying equipment requirements. The administrative domain level is where the administrator of the system intervenes by entering new policies, new attack counter-measure logics, and by taking decisions when needed for a counter-measure. There are two components in the administrative domain level. The Violation Detection analyses the monitoring information provided by the element level (and therefore from equipment at different locations in the network) and generates alerts when an attack is detected. The System Manager has a policy service for specifying, storing and deploying policies to distributed policy management agents (PMAs). The PMAs interpret the policies relevant to them. They subscribe to receive events which trigger the policies and take the relevant actions. For example a System Manager PMA can receive an attack notification from violation Detection and perform a response action on the firewall element or an anomaly notification may trigger a reconfiguration of the Violation Detect itself to adapt its detection strategy. Note that PMAs can be distributed in other components e.g. Violation Detection or even Firewall elements. All components of the architecture can be distributed so there can be one or more Violation Detection components and System Managers per administrative domain. This is summarised in Figure 2-1 below.

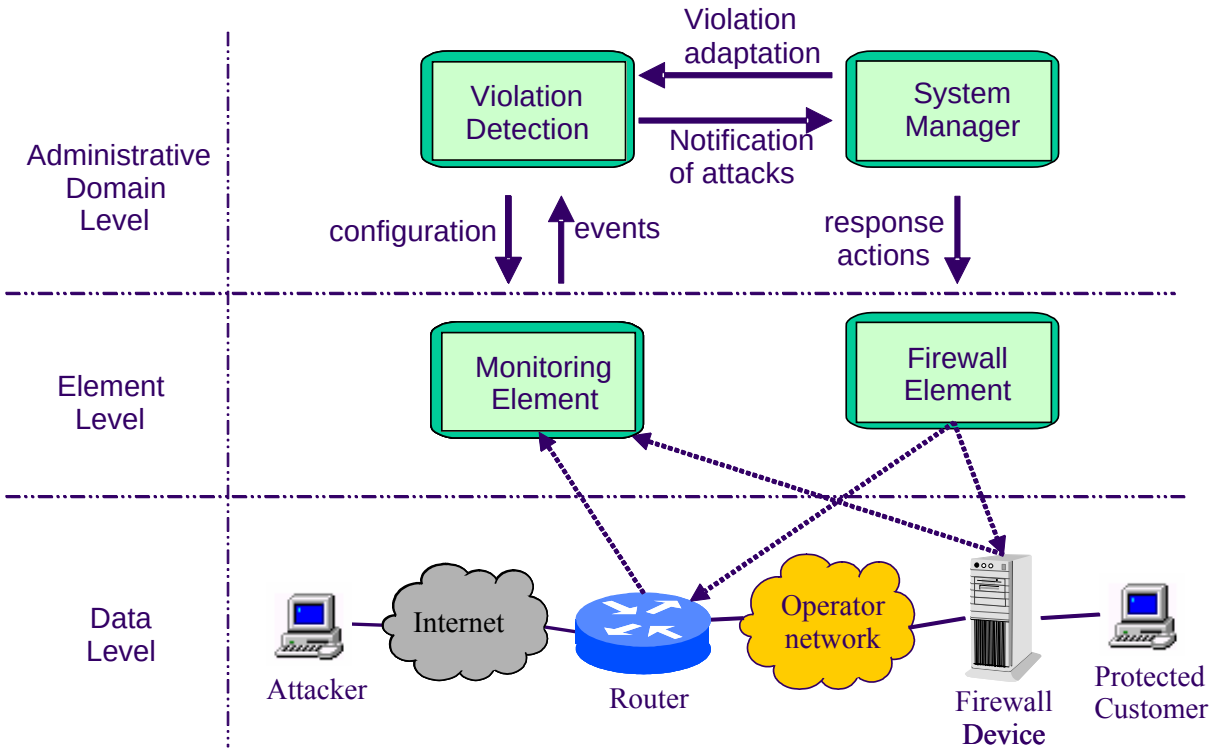

Figure 2-1: Overview of architecture

3 Devices

3.1 Diadem Device Types

We define a Device as a piece of equipment in the data plane (or data level) that can be accessed by a component of the element level for configuration purpose. It can be accessed for various reasons: to retrieve monitoring information, to change its configuration in order to enforce response policies or to monitor specific data flows, for instance.

In the Diadem Firewall project, we will use devices of four types:

- router
- open firewall
- open high-speed firewall
- commercial firewall

The Devices can each have their own configuration interface. They do not have to comply with a specific standard. Consequently, other types of devices can be included in the Diadem Firewall network, as long as the device can be configured via an external interface. It is the role of the Elements to act as an abstraction layer for the underlying devices.

A router device is used for monitoring the network traffic, using mechanisms such as Netflow or IPFIX. It can also be used in the reaction phase by installing router rules to block, shape, or re-route specific flows. The open firewall will be developed in the project. It will be possible to extend its capabilities, such as adding a new ALG (Application Level Gateway). The open high-speed firewall is an open firewall with a high performance FPGA-based hardware classifier. The commercial firewalls are integrated in the distributed architecture using the firewall element to control and configure the firewall's specific interface.

3.2 The open firewall

The open firewall is a firewall built around open-source technologies. It is based on a Linux PC architecture. We call it open also for the reason that it is possible to extend its functionalities, through the installation and execution of program modules which can adapt the behaviour of the firewall to perform a specific traffic analysis algorithm or to perform application level filtering of traffic etc...

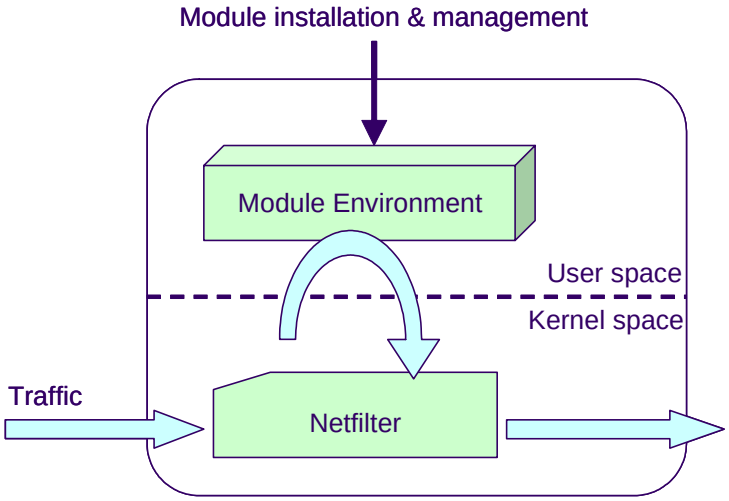

Figure 2-2: overview of the open firewall architecture

The minimum functionality of the open firewall is based on the standard functions of the Linux Netfilter which are:

- Access Control List (ACL), in order to drop IP flows
- re-routing of IP flows
- redirection of IP flows to user space

However, the case might arise that the core functionality described above are not sufficient. For instance, in the case of the TCP SYN flood attack described in D7 [3], it is not possible to filter out the attack traffic if the firewall is limited to layer 3-4 processing. Indeed at this level, all the attack traffic is perfectly valid. Therefore we need to extend the functionality of the open firewall in order to filter out efficiently the malicious packets. This is where the Module Environment is useful: the Firewall Element can install and manage a program module, which runs in user space on Linux, to perform required processing. The relevant IP flows can then be redirected to user-space to be processed by the module.

3.3 The open high-speed firewall

The open high-speed firewall has basically the same structure as the open firewall, with the exception that it is complemented with a high-speed classification engine. This Classification Engine (CE) takes over the rule-matching functionality and implements it in hardware. The basic architecture is shown in Figure 3-2. Here the CE is attached to the firewall element by a PCI-bus interface over which it receives control commands and packet data. The control commands are initiated through the Classifier API by upper layer software. The according functionality is described in detail in D6. The commands are used for CE setup in the initialization phase as well as for dynamic rule updates at runtime.

As the CE will be another filtering mechanism in the firewall architecture, the Linux kernel has to be extended with a loadable module. This module mainly handles two tasks:

- realization and communication of API commands to the CE and
- composition of a search key from the pertinent fields of the packet headers.

The basic architecture of this loadable module and how the module can be included into the Linux kernel will be covered in Section 3.3.3.

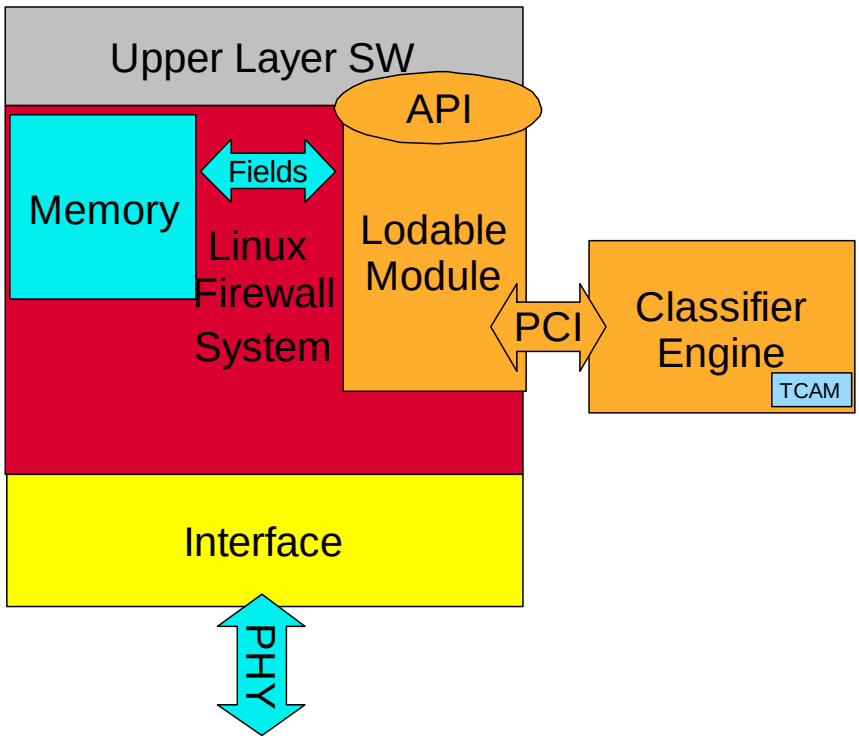

Figure 3-2: Basic architecture of the open high-speed firewall

3.3.1 The High-speed Classifier Engine

The high-speed Classifier Engine (CE) is designed to be used by a firewall device. It will be implemented on a PCI FPGA board. The purpose of the CE is to offload processing-intensive packet classification functions from the firewall device. Therefore, it implements a subset of the firewall-device functionality. The CE is configured and initialized by the firewall device.

The lookup tables, which are a set of packet-filter rules, are stored in fast static memory (SRAM). The multi-field search is currently performed by an associative memory in TCAM technology (ternary content-addressable memory). It can also be implemented using TCAM emulation in SRAM. For details on an implementation of the algorithms based on SRAM only, please see [9]. The supported functions include:

- Multi-field packet classification based on the parallel packet-classification algorithm P^2C [8,9,10].
- Single field searches based on the Balanced Routing Table Search (BaRT) algorithm [7].
- A classification result including the rule number (ruleID) and the action associated with the matching rule.

With these functions it is possible to support exact match, longest prefix match, and multidimensional range match searches on the CE. An example of the overall multi-field search algorithm is given in Figure 3-3.

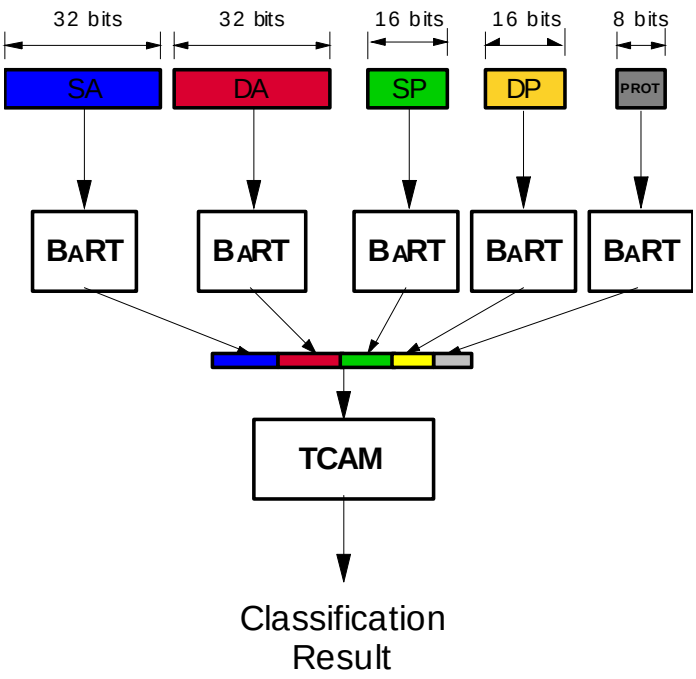

Figure 1-3: Parallel Packet Classification scheme (P^2C)

In this example the search has to be performed on a 5-tuple of fields which are extracted form the packet header: the 32 bit source address, the 32 bit destination address, the source port and destination port with 16 bits each and the transport protocol with 8 bits. For each field that is extracted from the packet header an independent field search is performed. The algorithm employed for this single-field search is the Balanced Routing Table Search (BaRT) [7]. According to the encoding mechanisms of the parallel packet-classification scheme P^2C an intermediate result vector is constructed. This intermediate result vector is the input to the multi-dimensional search which in this example is performed using a TCAM. For a more detailed description of the P^2C encoding scheme and the encoding styles for the intermediate result vector, please see [10].

3.3.2 Algorithms

3.3.2.1 Balanced Routing Table Search (BaRT)

BaRT [7] is a scheme for exact-match and prefix-match searches that achieves high search performance, suitable for 10 Gb/s link-speeds and beyond, by efficiently processing the search key in segments of about 8 bits. It requires only four memory accesses in the worst case to search a 32-bit IP address and two accesses for a 16-bit port number, all of which can be performed in a pipelined fashion.

3.3.2.2 Parallel Packet Classification (P^2C)

The P^2C scheme [10] employs the concept of independent field searches as shown in Figure 3-3. The word *parallel* emphasizes the parallelism available between the independent field searches. The key element of the scheme is a novel encoding of the intermediate search results which significantly reduces the storage requirements and minimizes the dependencies within the search structures, thus enabling fast incremental updates. The encoding involves several styles that can be applied simultaneously and allows several performance aspects to be balanced at the granularity of individual rules. The scheme can be used with a variety of search algorithms and memory technologies. In this project we will employ a configuration in which the fields are searched using the BART scheme [7] in SRAM and the multidimensional search is implemented using a TCAM emulated in SRAM [9].

3.3.3 Netfilter and the CE

The Linux kernel 2.6 defines its own filtering tool, called Netfilter. In this architecture, the user defines rules for how packets will be treated by the system. The basic elements of this architecture are packet filtering, network address translation (NAT), packet mangling, and manipulation. To facilitate this variety, Netfilter differentiates between input, output, and forwarding path. To supplement Netfilter, the user can add modules to the built-in solutions. Figure 3-4 shows the work of Netfilter and the packet handling with Linux kernel 2.6.

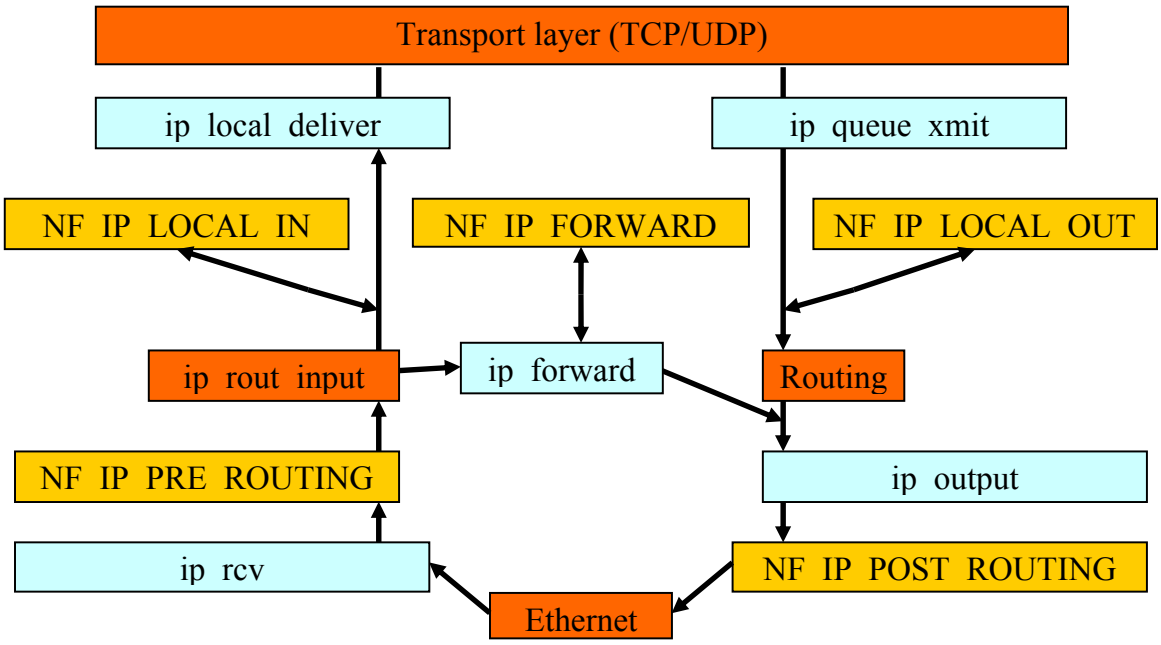

Figure 3-4: Linux Netfilter architecture

The Netfilter hooks are capitalized in Figure 3-4. Kernel modules can register to listen at any of these hooks. When a Netfilter hook is called from the kernel networking code, each module registered at that point is called, and can manipulate the packet. The module can then tell Netfilter what to do with the packet.

The usual process after receiving a packet from Ethernet is to deliver it to ip_rcv. It tries to find errors in the IP-header. If it cannot find any, ip_rcv calls NF_IP_PRE_ROUTING. This function is called a *Netfilter hook*. These hooks are used to dynamically manipulate the IP packets. This option was implemented to guarantee that the kernel and Netfilter do not influence each other too much. The hooks have a second advantage: It is possible to add further functions to Netfilter without changing the Kernel code.

When Netfilter finishes its work the packet is delivered to ip_route_input. This function decides whether the packet should be forwarded to another system or is targeted to the own system. If the packet's destination is the local system it will be passed on to ip_local_deliver. At this point the decision is made to which function within the transport layer the packet has to be delivered. Among other functions NF_IP_LOCAL_IN is called, which is another Netfilter hook. Afterwards ip_local_deliver_finish will complete the work. If the packet should be forwarded, ip_forward is called. Now the TTL is decreased and if the packet is too old it is dropped. After calling NF_IP_FORWARD the packet is passed to ip_output.

If the system itself wants to send a packet, ip_queue_xmit is called. This function calls NF_IP_LOCAL_OUT and determines the next hop the packet will take through the network (in Figure 3-4 it is called Routing). Then the packet is forwarded to ip_output. Among other functions NF_IP_POST_ROUTING is called and the packet delivered to the network.

The high-speed classifier engine will complement the built-in firewall architecture in Linux by adding further filtering functionality to the system. This is shown in Figure 3-5.

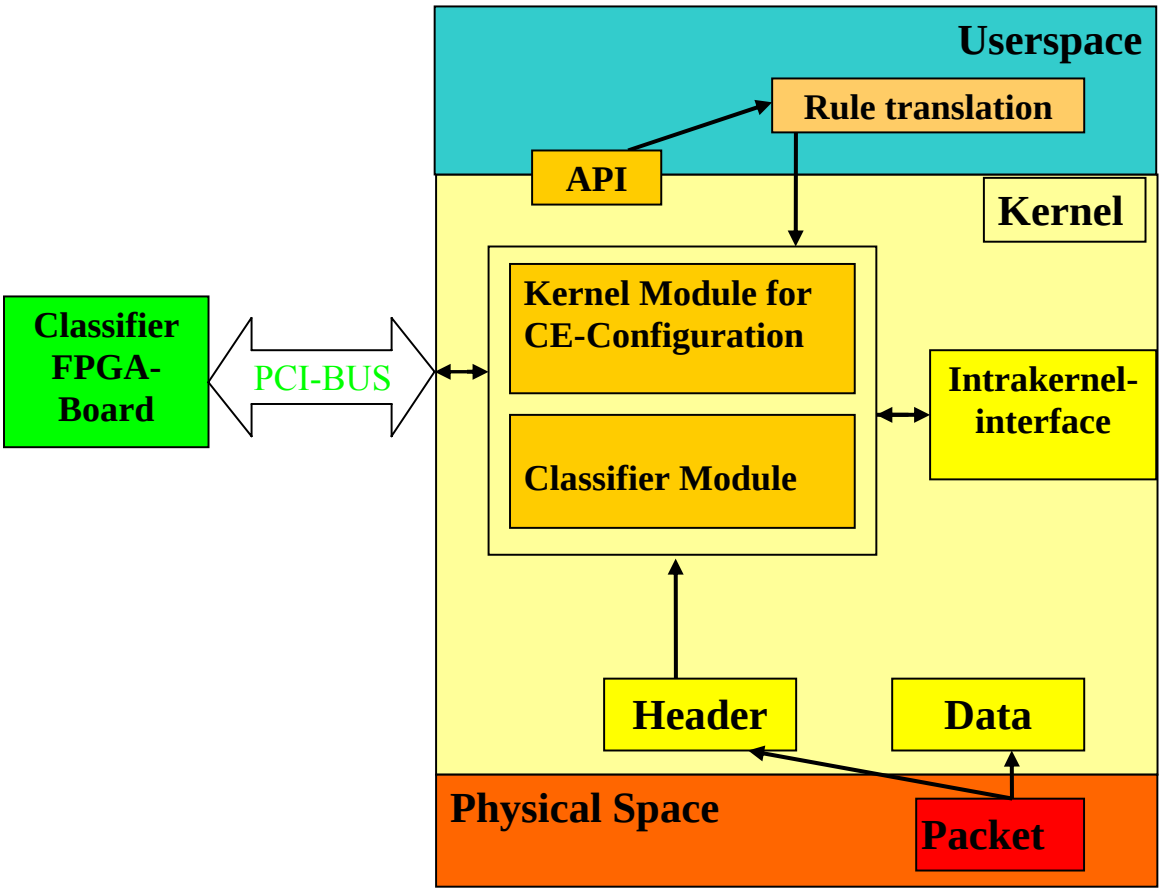

Figure 3-5: Kernel – Classifier Communication

An arriving packet contains payload as well as header information for the different protocol layers. These parts are separated by the Linux kernel and stored in an according data structure, called sk_buff (socket buffer). To realize the communication between the classifier engine and the Linux kernel it is necessary to implement a special Linux module to do this work. This module consists of two parts. The first part forwards header information to the classifier and receives the classification result. This module is called the Classifier Module. The second part configures the Classifier Engine by initializing and updating the packet-filter rules. This module is called Kernel Module for CE Configuration.

The Linux socket buffer stores the header fields in a data structure. This structure is passed to the Classifier Module, which forwards the pertinent header fields to the FPGA board across the PCI bus. Then it awaits an answer and passes on the classification result.

The Classifier API offers the user the possibility to define new rules for the classification. These rules have to be contained in a data structure that is passed to the Classifier API. These rules are now translated to expressions the CE understands and passed to the Kernel Module. These expressions are forwarded to the CE across the PCI bus. Now every packet can be classified according to this packet-filter rule-set.

In the Netfilter architecture there are three different hooks to which the Classifier Module can attach. This is shown in Figure 3-6.

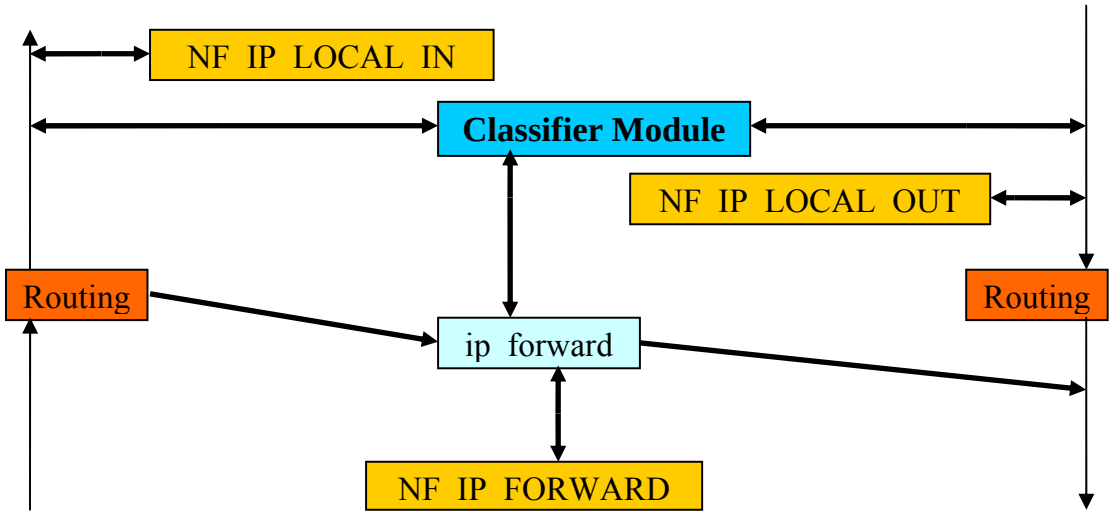

Figure 3-6: Classifier module in the Netfilter framework.

As an example for the forwarding of packets, Figure 3-7 shows where the Classifier Module will start to work. In this case, ip_route_input decided to forward the packed and passed it to Linux Netfilter. The Classifier Module will be hooked in this system before the regular Linux filtering starts.

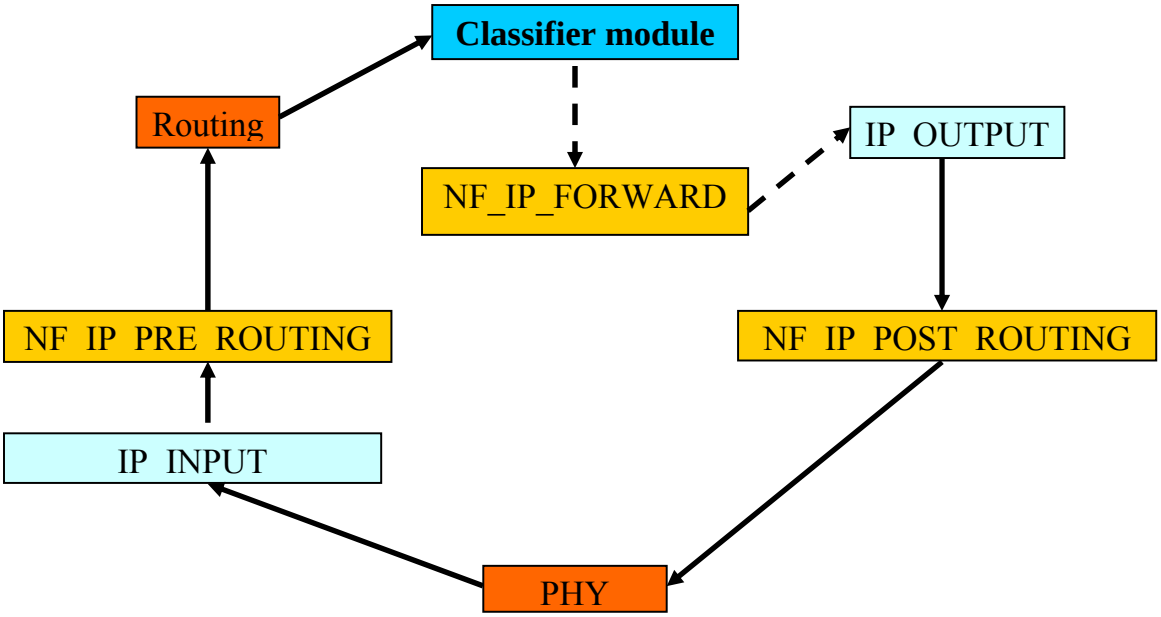

Figure 3-7: The Communication between Linux Netfilter and the Classifier Module

According to the packet filter rule set, the CE returns an action, which will be interpreted by the according kernel module. Only if the action was an *accept* the packet will be forwarded to the Linux build-in filter functions. As seen in Figure 3-7, Linux packet filtering can be used in addition. In this case, Netfilter would decide what to do with this packet if the classifier module decides to accept the packet. So the Classifier Module will have a kernel internal interface to communicate with other Linux kernel modules, but also an interface to communicate with the CE. This will be achieved across a PCI bus and an according PCI driver.

4 Monitoring Elements

This section describes the architecture and function of Monitoring Elements. First, the separation between Monitoring Elements and Monitoring Devices is explained. Then, subsection 4.2 summarizes the interaction between Monitoring Element and Violation Detection. Finally, a functional block diagram is provided.

4.1 Separation between Monitoring Elements and Monitoring Devices

The task of Monitoring Elements and Monitoring Devices is to monitor network traffic and deliver monitoring data to the Violation Detection system. While Monitoring Devices perform the actual monitoring, Monitoring Elements provide a common Monitoring API for device control and configuration as well as the IPFIX conform export of the monitoring data.
The separation between Monitoring Element and Monitoring Device may be explicit, e.g. if a router acts as a Monitoring Devices and exports statistics about ongoing IP flows. In this case, the associated Monitoring Element hides device-specific details by translating Monitoring API calls into device specific CLI commands and, if necessary, by converting the exported monitoring data into IPFIX format.

It is possible to associate more than one Monitoring Devices to a single Monitoring Element. This may be useful if the Monitoring Devices watch over different networks, if they support different kinds of monitoring mechanisms (e.g. IP flow metering and packet sampling), or for load balancing reasons.
However, the separation between Monitoring Devices and Monitoring Element may vanish, e.g. in case of an implementation that integrates the whole functionality in a single system.

4.2 Monitoring Elements and Violation Detection

Monitoring Elements expose a device-independent Monitoring API that allows to control and configure the associated Monitoring Devices in a standardized way. As described in [1], the Monitoring API is based on the Netconf protocol [4] and its operations. The Monitoring API is used by the Violation Detection system in order to dynamically adapt the monitoring processes according to its current needs and interests.
Monitoring Elements export monitoring data using the IPFIX protocol [5]. In terms of the IPFIX reference model, a Monitoring Element is an IPFIX device while the recipient of the monitoring data, i.e. the Violation Detection system, is called collector.
Figure 4-1 illustrates the interaction between Monitoring Element and Violation Detection.

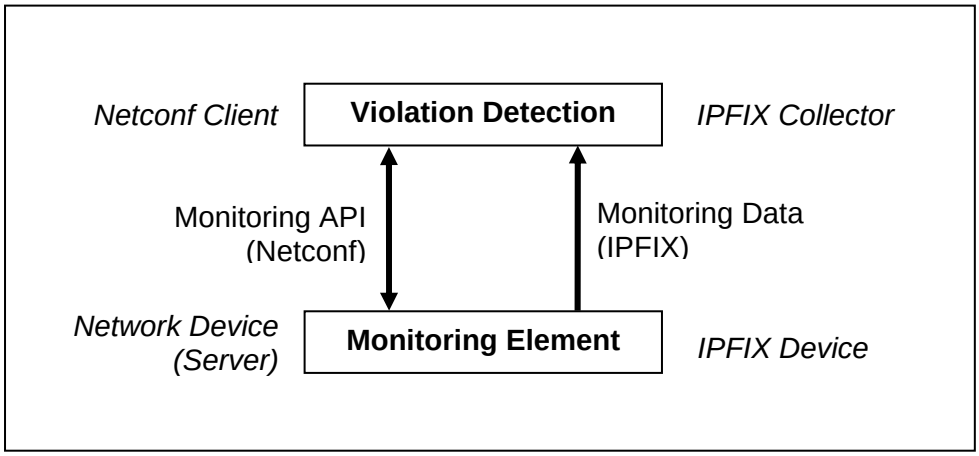

Figure 4-1: Interaction between Monitoring Element and Violation Detection

4.3 Architecture

4.3.1 Monitoring Element without Aggregation

The diagram in Figure 4-2 depicts the functional blocks of a Monitoring Element that does not integrate the aggregation functionality of an IPFIX concentrator. As mentioned in 4.1, the partitioning of the monitoring tasks between Monitoring Elements and Monitoring Devices may vary, i.e. some of the functional blocks may be situated in associated Monitoring Devices. Also, the actual implementation structure of the Monitoring Element may differ.

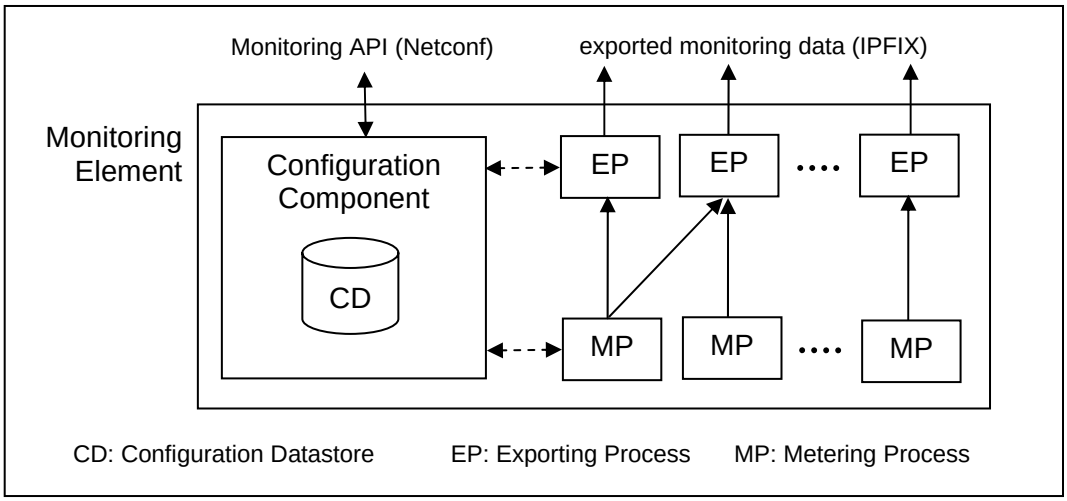

Figure 4-2: Monitoring Element without Aggregation

According to the abstract model of an IPFIX device as described in [6], the monitoring functionality of the Monitoring Element is divided into metering processes and exporting processes. A metering process gathers and optionally pre-processes monitoring data observed at an observation point (e.g. network interface). An exporting process transfers the monitoring data issued by one or more metering processes to the Violation Detection with help of the IPFIX protocol. Monitoring data consists of either IP flow records, sampled packets, or some other kind of information gathered by the metering processes.
Metering and exporting processes are configured and controlled by the configuration component. The configuration component also integrates the Netconf functionality and exports the Monitoring API. As specified in [4], the configuration data is stored in a configuration datastore. In a simple implementation, the configuration datastore could be realized with a configuration file. A more sophisticated configuration datastore stores more than one configuration, i.e. not only the current running configuration, but also candidate configurations that are to be verified before being enforced, start-up configurations etc.

4.3.2 Monitoring Element with Aggregation

Architecture differences if Monitoring Element supports Aggregation. As shown in Figure 4-3, aggregation processes can be included in the Monitoring Element. The provided aggregation functionality allows to compress the amount of Netflow accounting and IPFIX data, e.g. by summarizing statistics of flows directed to the same host or network. The input might come directly from a connected metering process or from one or more distributed IPFIX exporters.

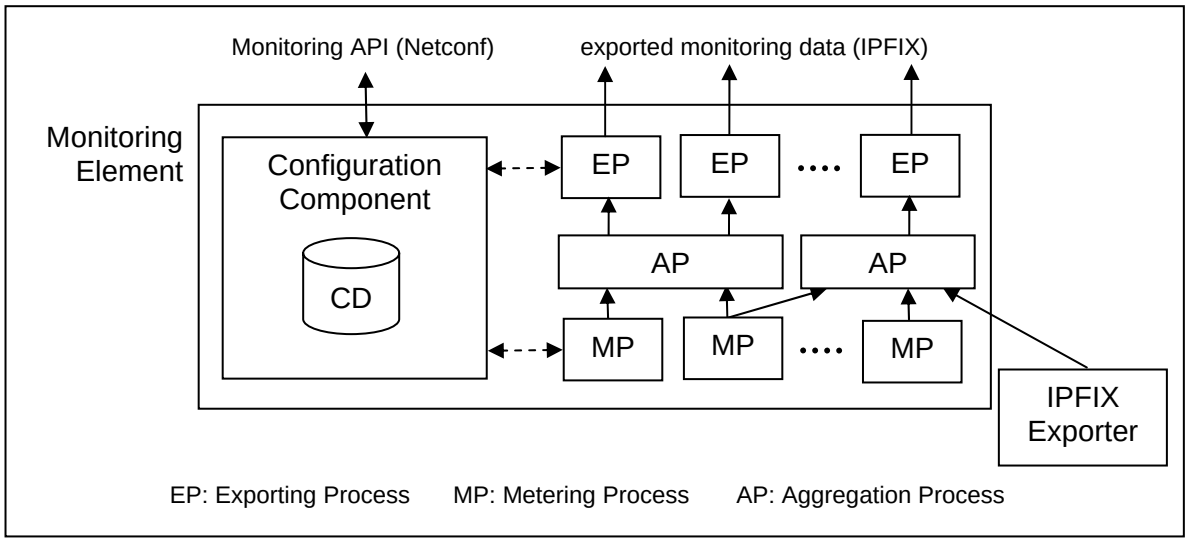

Figure 4-3: Monitoring Element with Aggregation

The performed aggregation is specified by aggregation rules. According to the IPFIX aggregation mechanisms (work in progress), some parameters are defined by implicit rules while others can be specified through configuration. The Violation Detection makes use of the Monitoring API to set the parameter of the aggregation processes according to the currently required granularity of IPFIX data. Typically, information about networks is sufficient for most statistic anomaly detections. Only after the detection of suspicious behaviour, the particularly involved networks should be monitored in full detail.

5 Firewall Elements

A Firewall Element (FE) is a control plane element that on one side interfaces with System Manager and on the other side controls a Firewall Device (FD) as presented in the Figure 2-1. The main task of the FE is to enable SM to control and manage FEs and attached devices with an aim to counter attacks detected in Violation Detection system.
The FE will be presented with functional requirements in section 5.1, and architecture in section 5.2

5.1 Functional requirements

A Firewall Element enables the System Manager to control and manage the Firewall Element and attached device(s). The FE abstracts the specific details of the various types of Firewall Devices identified in section 3.1, and provides a standard interface for accessing the attack response mechanisms (presented in [1]) on these devices. Essentially the FE hides the diverse functionality of the FD from the System Manager components. Response mechanisms as foreseen in DIADEM project are broader than those in traditional firewalls. Therefore the functionality of the firewall layer must enable for example access to the QoS mechanisms and routing table manipulation, etc. of the Firewall Devices. Since the response mechanisms can be extended, the firewall layer should also support such extensions.
In addition we provide the capability of implementing Policy Management Agents (PMA), which are really system management components, to interpret policies on the Firewall Element. Thus the System Manager can either directly invoke response actions or events can trigger policies in local PMAs to invoke the response action i.e. the Firewall Element can take local decisions about response actions if that is appropriate. This avoids the need for events to propagate through the Violation Detection and System Manager, if a local Monitoring Element in the Firewall Device detects a particular anomaly for which a local response action is appropriate. The Firewall Element may also support loading and running program modules to modify its functionality. This would be the way to install a local PMA but there might be a rather complex response action which contains a number of conditional sub actions that should really be implemented as a program module or script. The program module environment enables the response capabilities of the Firewall Devices to be dynamically modified if necessary.

There are four groups of management and control tasks related to the policies, modules, capabilities and configuration.

i) The optional PMA in a Firewall Element supports the capability of load, remove, enable and disable policies. The PMA interprets Response policies and performs internal actions in the Firewall Element or on the Firewall Device. The division of enforcement is optional; the FE can enforce a policy or a part of it if it not possible to enforce it on FD itself.

ii) The functionality of the FE and attached devices can be extended with program modules that run in module environment and perform the tasks that otherwise are not supported by attached device(s) or the element itself. The FE must be able to load, remove, stop and start program modules.
iii) Different Firewall Devices can have different response capabilities. The Firewall Element therefore must be able to report the capabilities of attached devices and program modules to the System Manager or other system entity. The capability list has to be updated dynamically when new devices or program modules are added or removed.
iv) The configuration of the element is related to the state of the element policies and code modules. The element must be able to report the status of the loaded policies and program modules. The element must be able to export or load an entire configuration as a single file.

The System Manager can directly invoke response mechanisms, so this requires that the response mechanisms are exported by FEs and can be accessed remotely. All the functionality of the FE needs to be accessible remotely over the internet. We are evaluating the possibility of implementing the APIs of the FE as web services for remote access.

5.2 Firewall Element architecture

The Firewall Element architecture is presented in Figure 5-1. The Firewall Element is located between the System Manager and the Firewall Device. It has a number of functional blocks called modules and it exports two interfaces to the System Manager.
The Service module exports the service API that enables the System Manager the tasks of control and management of the FE and FD. Through this API the System Manager can deploy code modules and response policies to the element and control the activation and deactivation of the deployed policies.
The PMA interprets policies and invokes response actions. It supports loading, removing, enabling and disabling of policies. It is also responsible for reporting the status of the policies in the element, so the System Manager can query the element for loaded policies and their status.
The Code module is responsible for loading and removing of the code modules and reporting their status to the Service module, so this information is available to the System Manager. The code modules run in the Module environment.

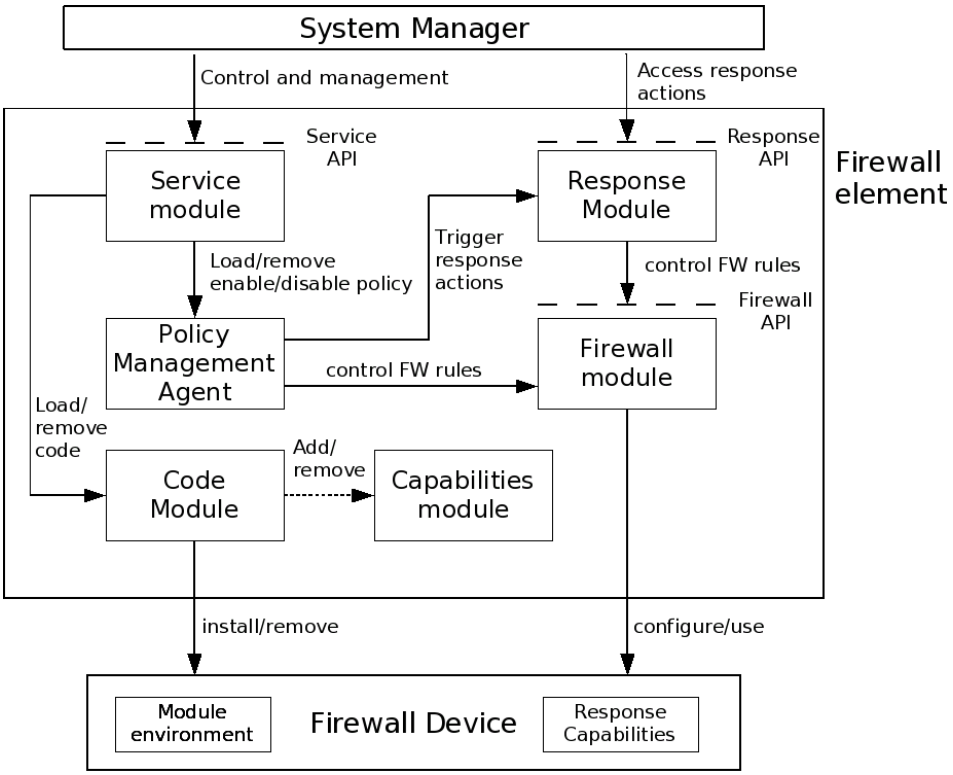

Figure 5-1: Firewall Element architecture

The Capabilities module obtains the initial set of capabilities of the Firewall Device and updates this when new program modules are added or removed. The Capability module reports the current set of available capabilities to the Service module which can be queried from the remote System Manager. The Response module exports the response API to the System Manager so it can access the response mechanisms of the Firewall Element and Device directly. The Response module uses Firewall API to access the Firewall Device response mechanisms.
The Firewall module exports internal Firewall API that is accessed by the Response module and enables control of the firewall rules enforced on the FD. The Firewall API implements device dependent communication with the FD and hence hides the device dependencies of the various different types of FD. All three mentioned APIs in this section are further explained in [1].

6 The Violation Detection

6.1 General Objectives

Diadem firewall deliverable D3 [13], section 5 provides the set of requirements for the definition of the violation detection architecture. These requirements cover 4 different areas:

- Relationships between the monitoring system and the violation detection system.
- The lower level violation detection system that is able using monitoring results to detect attacks.
- The upper level violation detection system that is able using attack detection results to aggregate and/or correlate these results to provide the system manager with a synthetic view.
- Relationships between the violation detection system and the system manager.

These requirements allow us to define general objectives for the violation detection facility architecture:

- **Modularity**. As explained in D3 [13], attacks in general and DoS attacks in particular usually include several phases. These phases usually use different attack techniques and cannot be detected using a single detection technique. As a result it is important that the detection system is not specialised for a specific phases or type of attack and also needs to be adaptive. This means that different detection technique have to be used concurrently as a part of the detection system. As the project does not have the ability to develop tools for all existing detection techniques, it is important that the architecture permits existing detection tools to be integrated in the detection system with a limited amount of development. As attack and detection tools are expected to evolve widely in the future such modularity will also permit the integration of future detection tools without loosing existing capabilities.
- **Distribution**. Due to the nature of DoS attacks that can generate large amounts of traffic, and to the nature of some detection systems (e.g. signature based IDS) that require large amounts of information to be transported between the monitoring system and the detection system, using a fully centralised system are usually impossible. This impossibility can result from throughput limitations between the monitoring system and the detection system, from performance restrictions on the detection system or from financial restrictions associated with the cost of transporting large amounts of data over large distances. For example, in the case of signature based IDS such as snort, if the amount of traffic passed to the detection module is not properly restricted, it is preferable to locate the monitoring element and the lower level detection element on the same physical device.
  Some attacks need a faster detection (for example because a prompt reaction is needed) while others do not require such urgent reaction. As the latency needed in order to detect an attack is not known before the attack is analysed, it might be useful to locate detection modules closer to the traffic related to the attack in order to avoid network latencies.
  Another benefit from using a distributed architecture is to limit the impact generated by the failure of a detection component on other detection components. In the case of a failure a centralised architecture might prevent any kind of analysis for the traffic of a whole network as similar detection components might be physically replicated in a distributed architecture, the failure of a component only impacts the traffic/attacks monitored by such component.
  Because detection components can be developed on distinct devices, resources allocation and control between heterogeneous detection processes is simplified. This reduces the complexity of the violation detection architecture.
  These restrictions apply to aggregation/correlation functions for the same reasons. As a result the detection system can be seen as a hierarchical architecture where lower level detection components constitute the leaves of the tree and aggregation/correlation components constitute the inner nodes.
- **Open Interfaces**. The relation between detection and aggregation components is similar in nature to the relation between the system manager and the detection system:
  - Aggregation/correlation components receive notifications from lower level detection components.
  - The system manager receives notifications from aggregation/correlation components.

  As a result similar interfaces should be used between these entities for notifications. This will facilitate the treatment of alarms within the system manager/aggregation components and reduce development work. Moreover as a large number of existing violation detection tools as well as upcoming detection tools generate messages using the IDMEF format, the communication between these entities should use this format.

6.2 Overall Architecture

As represented in Figure 6-1, the violation detection facility is organised in several modules:

- The IPFIX collector receives monitoring data from monitoring elements.
- Detection modules. Detection modules apply detection methods on monitoring data. Detection results are then passed as IDMEF events to the PMA. Existing detection tools (such as snort) may be integrated to the VDF as detection modules.

- The Policy Management Agent (PMA) stores and interprets detection policies which may be triggered by IDMEF events or can be used to filter events. Possible actions are notify, log, store, defer an investigation module, or discard an event. Other actions might be to reconfigure the monitoring elements
- The Event Database. Interesting IDMEF events can be stored here for later investigation and correlation with other events.

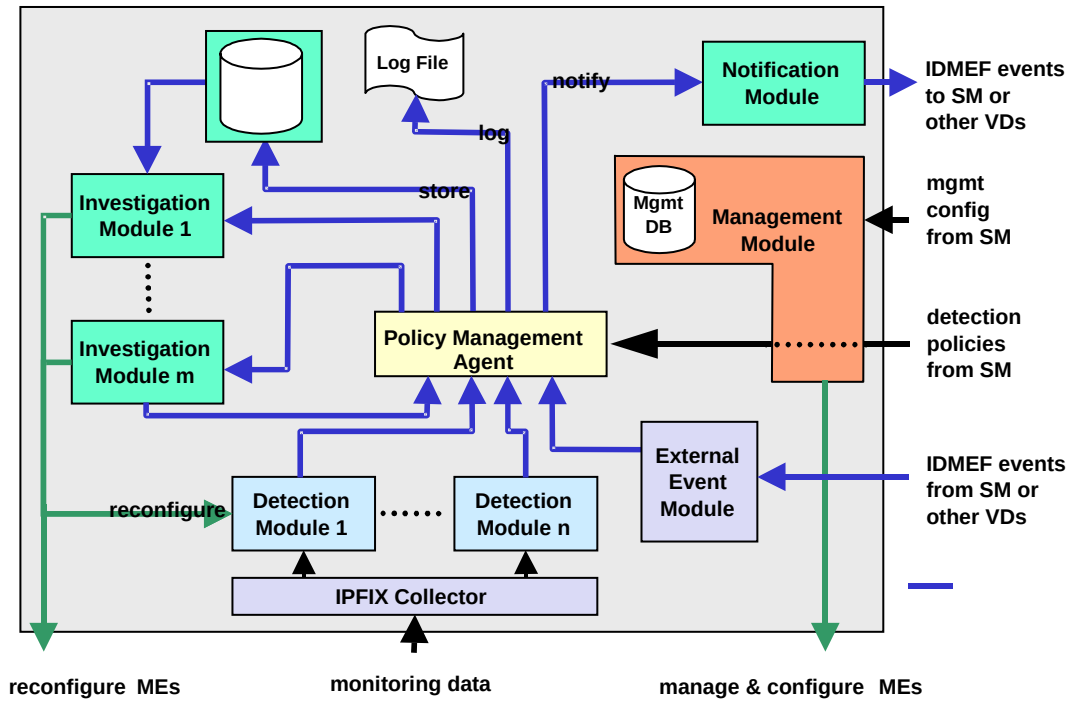

Figure 6-1: Overall Violation Detection Architecture.

- The Investigation Modules are used to examine events that cannot be clearly classified as an attack or not. Possible investigation methods include:
  - compare/correlate event with earlier events stored in event database
  - reconfigure detection modules and/or MEs (management module controls access to detection modules and MEs)
    - new detection results passed as IDMEF events to the PMA
    - either these events can be clearly classified as an attack or they are also passed to an investigation module

  If the investigation module concludes that an event is an attack, an IDMEF event is passed to the PMA
- The Notification Module sends IDMEF events (notifications) to the SM or other VDs. The list of recipient VDs is managed by management module
- The External Event Module constitutes the counterpart of the notification module. It receives IDMEF events (notifications) from the SM and other VDs and controls which events are passed to Policy Management agent (via list of trusted VDs)
- The Management Module exports a management and configuration interface to the SM (Service API). It provides start-up configuration to all modules in the VD and to the MEs, maintains management data in management database and controls reconfiguration of detection modules and MEs via investigation modules.

6.3 The IPFIX collector

The IPFIX collector receives monitoring data from the Monitoring Elements and passes it on to detection modules. Monitoring data consists of IP flow records, sampled packets, or other kind of traffic information. According to the IPFIX protocol [5], Monitoring Elements export monitoring data using configurable templates. The templates in use have to be sent to the IPFIX collector periodically in order to allow interpreting and decapsulating the monitoring data correctly.

6.4 Detection Modules

6.4.1 TCP SYN Flooding Detection Module

Please refer to document D7 [3] for a description of the TCP SYN flood attack. The detection module collects IPFIX records concerning the destination IP address of the potential victim. It performs an anomaly detection of attacks based on the number of TCP SYN packets sent to the victim. The detection module is given a model for a normal number of TCP SYN packets in function of the time and it continuously performs a comparison between the monitored traffic and the model for normal traffic. When both differ by a certain threshold, an IDMEF event is generated and sent to the PMA. The details of the anomaly detection (algorithm, parameters taken into account, configuration values) will be determined during the development phase of the project.

6.4.2 Web Server Overloading Detection Module

The general behaviour of the Web Server Overloading Detection Module is to build from network-level measurement parameters provided by the IPFIX collector a view of application level actions. These inferred application level actions are then used to detect overloading attacks by matching detected results with a detection model. When an attack is detected, a notification is sent to the PMA in the violation detection facility.

The Server Overloading Detection Module is organised around four components as represented in figure 6-2:

- The parameters extraction element extracts useful measurement parameters from IPFIX records provided by the IPFIX collector. This module is also used to select among IPFIX records those that are of interest to the violation detection module. This is performed through the definition of monitored targets. Monitored targets include potential victims (web server addresses and ports) as well potential attackers (addresses/prefixes and ports/port ranges). The definition of monitored targets is included in the target database. Although this function may seem redundant this filtering function is aimed at making sure that records that are treated by other elements within the violation detection module are consistent with internal databases.
- The parameter estimation element uses extracted parameters to generate measurement parameters that are adapted to the inference engine. For example IP addresses may be transformed into country codes in order to limit the size of the inference model. Each parameter may need a different form of correction or adaptation. The estimation function is guided for each estimation element by an estimation model selected from the estimation database based on the type of parameter as well as the targeted web server. In the case of the IP address to country code transformation, this model would be a data structure representing relationships between these two parameters.

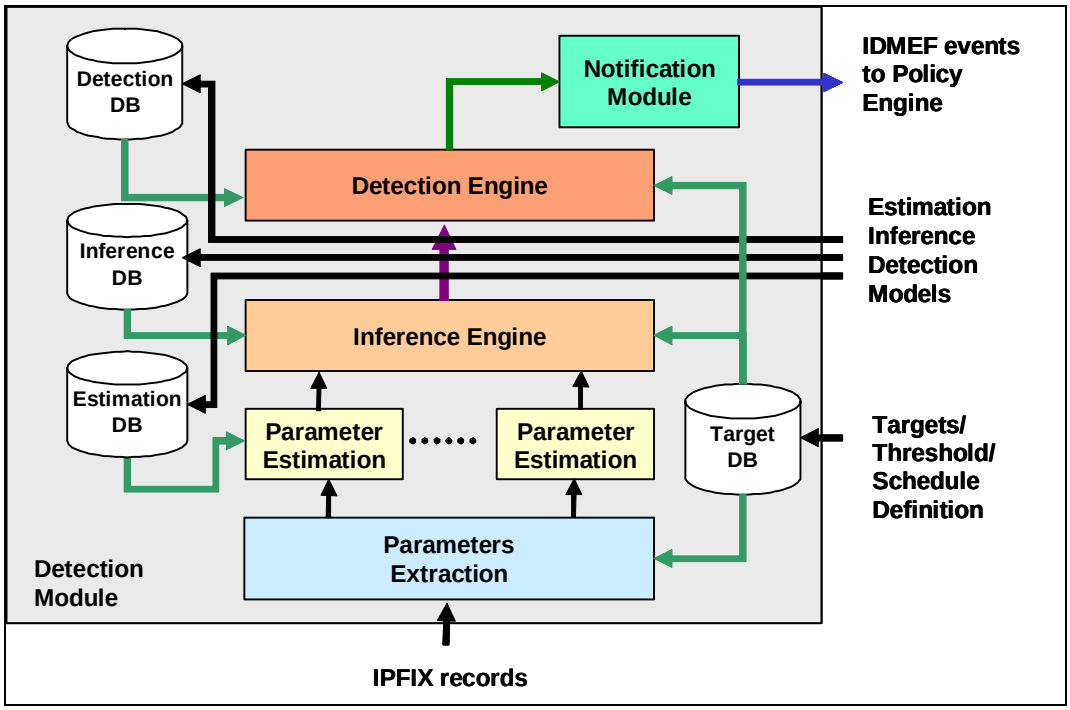

Figure 6-2: Server Overloading Detection Module.

- The Inference Engine is used to infer application level operations from estimated/corrected measurements parameters. For each set of estimated element, the Inference Engine will select according to the targeted server (defined by the destination address) as well as the target definition an inference model from the inference database. This inference model will be used along with estimated parameter to produce a tuple including the identity of the user (IP address) the action performed (HTTP header values) and the targeted web server. Although inference models are target specific, we separate the definition of targets from the definition of models for efficiency reasons. The definition of targets may change frequently (several times an hour) while the modification of models should happen less frequently (Once a week).
- The Detection Engine is used to check if the tuple received from the Inference Engine is compliant with the detection model. To do so, the Detection Engine selects the appropriate detection model from the detection database using the targeted web server information as well as the targets definition. This detection model is composed of two elements. A part of the model represents the normal behaviour of users for a specific server. The other element represents the acceptable deviation from a normal behaviour for that server. This threshold is target specific and provided in the target database. Periodically, the detection model is used along with the identity of the source and the action to decide whether an overloading attack is going on. The periodicity is described by the schedule associated with the target. If an attack is detected, an alert is sent to the notification module. This module produces an IDMEF compliant notification that is sent to the PMA. This notification includes the identity of the victim (address and port), the identity of the attacker (address and port), the type of attack, the level of deviation between the attacker behaviour and a normal behaviour, the number of packets/bytes related to the communication.

6.4.3 Traffic Volume Detection Module

The traffic volume detection module does not provide any detection capability per se. It acts as an interface or a translator between the monitoring system and the violation detection system. One of the goals of the traffic volume detection module is to report the amount of traffic associated with a particular signature as an IDMEF event to the PMA. This type of event can be used by investigation modules such as the tracking module to identify the sources of a DoS attack. Signatures are defined similarly to filters used by monitoring elements. When sampling is used by monitoring elements, the module is either able to infer the true amount of traffic or a range in which that amount is located. When other conditions (loss/filtering/rate limiting) may have impacted the measure, these conditions should be taken into account to evaluate the volume reported to the PMA. Information reported to the PMA includes the amount of traffic measured, the time range over which the measure was performed, the location (address, interface) where the measure was performed, an indication whether the volume of traffic was inferred or directly measured.

6.5 Policy Management Agent

The PMA receives policies from the system manager of the form

```
On event do action when condition
```

The policy can be used to process events. The `condition` field is used to filter out the IDMEF events of interest by comparing IDMEF fields to given values. The `action` field can represent the following

- notify: send event as IDMEF message to the SM or other VDs.
- log: write event to a log file.
- store: store the event in the event database.
- defer: pass the event to an investigation module for further examinations.
- discard: ignore the event, stop looking for more matching policy rules

Other policies can be used to make local decisions to modify the behaviour of the monitoring elements rather than only relying on this being done by remote System managers.

6.6 Investigation Modules

The following modules are examples of Investigation Modules that will be developed in the violation detection facility.

6.6.1 Aggregation Modules

Aggregation modules are used when distributed detection modules monitor the traffic for a given attack. In this configuration, the aggregation module identifies distributed attacks. In such a case each module may monitor a traffic that is not sufficient by itself to be identified as an attack but the traffic resulting of the combination of all traffics can be identified as an attack.
The aggregation module is organized around three main elements. The module receives IDMEF notifications from the PMA through the notification module. These notifications are stored in a local event database. The detection function retrieves periodically events from the event database. The periodicity is defined through the schedule provided in the target database by the system manager. The detection function then uses an aggregation function allowing the various deviations to be summarized through a single value. Note that this operation is specific to the detection scheme from which notifications are received (this operation is not meant to compare notifications of different nature).For example in the case of a detection function based on the volume of traffic, the aggregation function may be implemented through a simple sum of the volumes measured by each detection module. The aggregated value is compared to the threshold to decide if the attack is still going on. The result of the detection function is used to generate a new notification including the identity of the target, the level of deviation from the threshold, the volume of traffic associated with the attack. The notification is then sent back to the PMA as an IDMEF event.

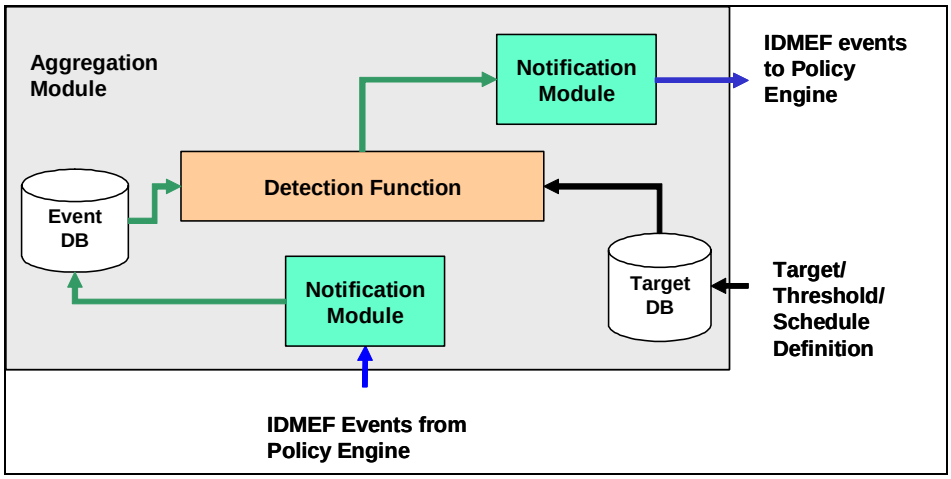

Figure 6-3: Aggregation module architecture.

6.6.2 Tracking Module

The following tracking module is just an example about how investigation modules might be used to provide tracking functions. A simple implementation of the tracking module could make use of the traffic volume detection module to identify the largest contributors for an attack. The tracking module is made of three main components:

- The collection function acts as a co-ordinator. Once a target and threshold are defined, the collection function uses the election function to define what node should be used to get information about the attack. Typically a node will include several interfaces that may be monitored. These interfaces as well as detection modules are then configured by the configuration function through the management module. Once events are received from the detection module, the collection function stores these events until events for each interface are received. These events are used to identify the interfaces that provide the largest contribution to the attack. The election function is then used for each of these interfaces to identify nodes where further investigations should be performed. The operation is then repeated. Once measurement results are received from nodes/interfaces closer to the attacker, nodes/interfaces that are closer to the victim are reconfigured so that flows are only accounted once. The tracking function stops when either one of these conditions no longer holds.
    - The attack is going on. This event is reported to the tracking module by the configuration of its target.
    - Nodes identified by the election function are in the local administrative domain and are not end nodes.
    - Flows associated with received events carry a volume of traffic that is over a threshold.

  In this case, the identity of the contributing nodes is reported to the PMA through an IDMEF event.

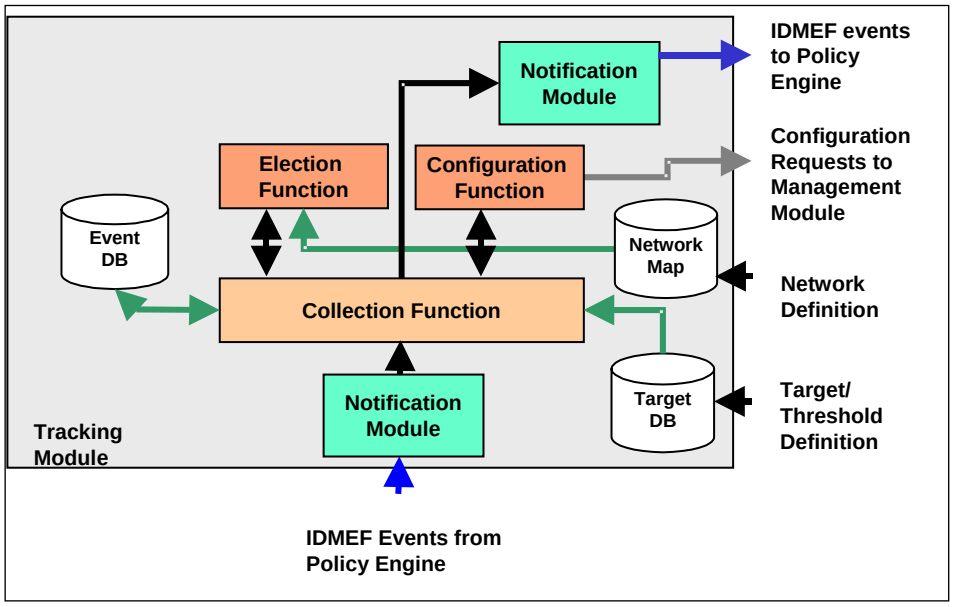

Figure 6-4.: Tracking Module architecture.

- The election function identifies the appropriate measurement nodes that should be reconfigured to track an attack. In order to do so, the election function uses a map of the network. This map includes the identity of potential monitoring nodes as well as their location on the network. Given monitoring interfaces, the election function is able to identify the nodes that contribute to the attack. These nodes may not be physically adjacent to the interface considered. When possible, the election function will provide the identity of each interface belonging to the node identified.
- The configuration function is used by the collection function to generate, enable and disable monitoring functions on measurement elements through the management module. It is also used to configure detection modules accordingly. To do so, the configuration function takes as an input an interface/node identity, and the target definition and produces a configuration request for the management module.

6.7 Management Module

The management module exposes the Service API to the System Manager which allows managing and configuring the Violation Detection and its modules. In particular, it allows adding and deleting detection policies in the detection policy repository of the PMA. The System Manager may also provide configuration for the monitoring environment which is passed on to the associated Monitoring Elements. Management data is stored in a management database.

7 The System Manager

This section describes the System Manager (SM) architecture and how it communicates with other modules in the Diadem Firewall architecture in Figure 2-1. The System Manager is not a single components but a set of components.

7.1 Overview

A system administrator specifies policies which defines reactions to events. Policies are of the form

```
On event (event parameters)
subject do action on target when condition
```

Policies are interpreted by Policy Management Agents (PMAs). The subject identifies which PMA is responsible for interpreting the policy. The policies are loaded into the relevant PMAs which subscribe to receive the events defined in the policies. When the event occurs, the policy is triggered and the PMA invokes the action if the condition is true. An action may be implemented locally with the element running the PMA or it may be a remote invocation across the internet on a Firewall Element (FE) or Violation Detection component (VD). The VD generates notifications i.e. events which triggers policies but the System Manager may also generate new events to trigger policies.

The System Manager consists of a policy service to support specification, storage and dissemination of policies to the distributed PMAs which actually interpret the policies. PMA can be implemented with VD components, FEs or as separate system manager components. This implies the system can be completely distributed or could form a hierarchical structure. For example, a number of VD components pass events to a centralised SM component which has a consistent logical view of an administrative domain so can make a decision based on this view. It could generate events which trigger policies in lower level SM components or PMAs in FEs which perform actions to reconfigure firewall devices.

The SM receives alerts/notifications from the VD. A notification indicates the type of attack, the target of the attack, the source of the attack (this may be spoofed), the signature of the attack etc. The SM uses this information to determine the appropriate response to the attack. This decision making takes place in PMAs and is specified in the form of policies.

Each PMA has multiple policies that can be loaded and enabled. Each policy is triggered by a specific event and defines an action to responds to the detected attack. The response policies equate to one of the following:

- Perform a response action on a Firewall Element as defined in [1]
  - For example, disconnect a connection, redirect packets, perform rate limiting etc.
  - The SM may enable or disable policies within the PMA in a FE to change its behaviour.

- Perform an action on the Violation Detection component
  - For example, to reconfigure the VD to look for a more specific attack signature.
  - Or to reconfigure the Monitoring Element to collate information pertaining to a specific host (via the VD).
  - The SM may enable or disable policies within the PMA in a VD to change its behaviour.

- Generate an event that is subscribed to by a PMA in a Firewall Element
  - This can be used to trigger an action in multiple Firewall Elements that perform the action in slightly different ways. For example, both a firewall and a router are Firewall Devices that can perform redirection; with the firewall a filter rule can be added that forwards the packet(s) to a different location or with the router by manipulating its routing tables.

- Generate an event that is subscribed to by a PMA in a Violation Detection component
  - Actions on multiple VD components containing PMAs to do reconfiguration may be triggered by an event generated by the SM rather than it invoking operations on the VD directly.

Figure 7-1 shows the overall architecture of the SM but not all the above interactions are shown to simplify the diagram.

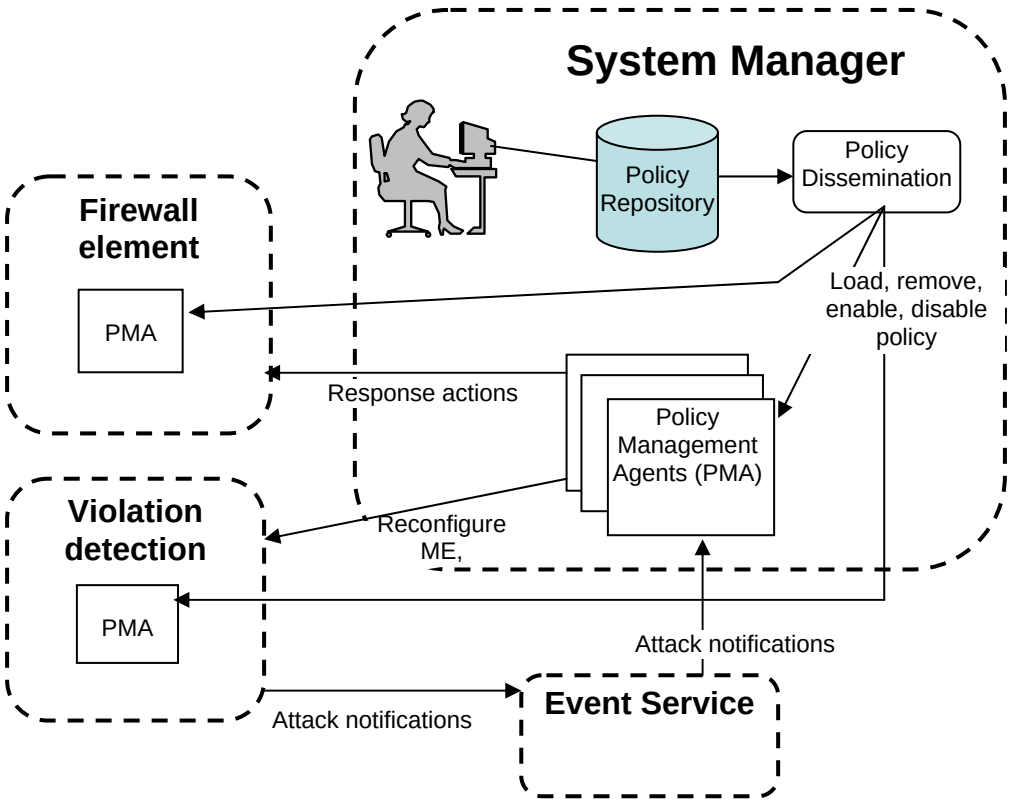

Figure 7-1: Overview of System Manager

7.2 Policy Manager Agent (PMA)

As stated above, the obligation policies in the form of event-condition-action rules area enforced by PMAs. The policies are compiled, loaded and enabled in the PMA which is essentially a policy interpreter. The policy specifies the action to be performed, making use of any of the parameters provided by the event notification.

PMAs register with an event service to receive relevant events generated from the managed objects of the system. On receiving an event, the PM queries a domain service to determine the target objects on which to perform the policy action(s). A policy can optionally specify constraints (conditions) which must evaluate to true before the action can be performed

7.3 Event Service

We are proposing to use the Elvin Event dissemination service. Events are published to the event router which forwards the event to any element which subscribes to receive the events. Subscription can be based on information contained in the event – source of the event, type of event message or severity of the warning in the event. For example a PMA in a particular FE may subscribe to receive events from a nearby VD component. A particular SM may only receive notifications when the severity of the attack on a web server is above a specified threshold.

7.4 Web Server Attack Use-case

This section describes the use of the SM to respond to a Web Server attack as described by Use-case 2, in Deliverable D7 [3].

The VD has detected a web server attack, for example from anomaly detection. The initial response by the SM is to reconfigure the VD. The policy to do this is shown in Figure 7-2.

```
inst oblig /SMPolicies/anomalyBasedAlert {
        on webAttackAlerts(attack_signature);
        subject /PolicyManager/SMManager;
        target t = /PolicyManager/VDManager;
        do t.reconfigure(configuration);
}
```

Figure 7-2: Reconfigure VD policy.

The policy states that there is a ***web attack alerts*** occurring, for example based on anomaly detection. The SM inspects the ***attack_signature*** and decides to ***reconfigure*** the VD with a new ***configuration*** to look for a more specific attack signature, in order to reduce the number of alerts or to be more certain that we do not have false positive alerts. As a consequence, the VD also reconfigures the Monitoring Element (ME) to provide the required information.

If after reconfiguration the SM receives a web attack event, where the source address is spoofed, then the SM can request the VD to perform a traceback action. The policy is shown in Figure 7-3.

```
inst oblig /SMPolicies/signatureBasedAlert {
        on webAttack(attack_signature, spoofed);
        subject /PolicyManager/SMManager;
        target t = /PolicyManager/VDManager;
        do t.traceback(attack_signature);
        when spoofed = = true;
}
```

Figure 7-3: Trigger traceback policy.

Upon receiving a ***web attack*** notification, if the source address is ***spoofed***, the SM triggers ***traceback*** in the VD. The traceback should provide all the Firewall Elements closest to the attackers based on the attack signature. This means the VD will generate an alert, with the list of elements, when the traceback is complete. The VD is likely to require the assistance of the Monitoring Elements for the traceback, so will have a policy enabled that is similar to that shown in Figure 7-4.

```
inst oblig /VDPolicies/tracebackPolicy {
        on traceback(attack_signature);
        subject /PolicyManager/VDManager;
        target t = /MonitoringElements;
        do t.reconfigure(configuration);
}
```

Figure 7-4: Reconfigure Monitoring Element policy.

Once the VD generates an event that signifies the completion of the traceback function, the SM can employ the Firewall Elements found to respond to the attack. An example of such a response is shown in Figure 7-5.

```
inst oblig /SMPolicies/elementsFoundPolicy {
        on elementsFound(attack_signature, list_of_elements);
        subject /PolicyManager/SMManager;
        target t = /FirewallElements/ToRateLimit;
        do addElements(list_of_elements, t) ->
            t.rateLimit(source, destination, threshold);
}
```

Figure 7-5: Enable rate limit policy.

Once the SM has been notified of the ***elements found*** to be closest to the attacker, it means the attack is real and specific, so the SM can instruct the devices to ***rate limit*** the traffic from the ***source*** and/or going to a ***destination***, obtained from the ***attack_signature***. The ***list of elements*** found is first added to the domain of elements to be rate limited, and then each element is configured to rate limit traffic to a predetermined ***threshold***.

If the traffic rate should fall ***below a threshold*** the rate limiting can be ***disabled***, as shown in Figure 7-6.

```
int oblig /SMPolicies/trafficRatePolicy {
        on trafficBelowTreshold(current_rate);
        subject /PolicyManager/SMManager;
        target t = /FirewallElements/ToRateLimit;
        do t.disableRateLimiting( );
}
```

Figure 7-6: Disable rate limit policy.

8 Conclusion

We have defined five high-level components: Devices, Monitoring Element, Firewall Element, Violation Detection, and System Manager. We define Devices as network equipment of the data plane, which can be controlled by a component of the Diadem Firewall architecture. We have identified four types of Devices that we will include in the project testbed: commercial router, open firewall, open high-speed firewall, and commercial firewall. The open firewall Device and its high-speed counter-part will be built during the course of the project, and they are described in this document. Monitoring Elements configure the monitoring functions of the Devices (e.g. Netflow for a Cisco router device) and collect the monitoring data issued by these devices. They aggregate the collected data and send it to the Violation Detection for analysis. Monitoring Elements are themselves configured by the Violation Detection (VD). The VD contains the attack detection logic. When an attack is detected, the VD sends a report to the System Manager through the Notification API (defined in the document D6). The System Manager analyses these reports and issues response policies to the Firewall Elements. The latter then interpret the response policies and convert them into configuration rules that are enforced on the appropriate Devices.

This document will be used as a reference to coordinate the developments made by the partners, and at a later stage for the integration of the prototypes implemented.

9 References

[1] DIADEM deliverable D4, Response Requirements Specification, July 2005
[2] DIADEM deliverable D6, Revised Interfaces Specification, January 2005
[3] DIADEM deliverable D7, Initial Demonstrator Specifications, January 2005
[4] R. Enns, "NETCONF Configuration Protocol", draft-ietf-netconf-prot-04, October 2004
[5] B. Claise, "IPFIX Protocol Specification", draft-ietf-ipfix-protocol-05, August 2004
[6] J. Quittek et al., "Requirements for IP Flow Information Export (IPFIX)", RFC 3917, October 2004
[7] Jan van Lunteren, "Searching Very Large Routing Tables in Wide Embedded Memory", Globecom 2001, San Antonio, TX, Nov. 2001.
[8] Jan van Lunteren, A.P.J. Engbersen, "Multi-Field Packet Classification Using Ternary CAM", IEE Electron. Lett. 38(1), (2002) 21-23.
[9] A.P.J. Engbersen, J. Van Lunteren, "Prefix-based Parallel Packet Classification", IBM Research Report, RZ 3210, March 2000.
[10] Jan van Lunteren, A.P.J. Engbersen, "Fast and Scalable Packet Classification", IEEE Journal on Selected Areas in Communications, vol. 21, No. 4, May 2003
[11] D.B. Chapman, E.D. Zwicky, "Building Internet Firewalls", O'Reilly, CA, 2000.
[12] P. Gupta, N. McKeown, "Packet Classification on Multiple Fields", Comput. Commun. Rev., vol. 29, pp 147-160, Oct. 1999.
[13] DIADEM deliverable D3, Attack Requirements Specification, July 2005