Distributed Network Security

Dipl.-Ing. Oliver Welter, Dipl.-Ing. Andreas Pilz
Technische Universitaet Muenchen
welter@tum.de

Abstract

IP-based networks form the base of todays communication infrastructure. The interconnection of formerly isolated networks brings up severe security issues. The standard approach, to protect the own network from abuse, is the usage of filter mechanisms at the border to the foreign network.

The raising complexity of protocols and the use of encryption techniques render most of these border-oriented systems useless, as their are not able to track or analyze the transfered data.

The approach discussed in this article splits into three parts – first we invent distributed sensors which enlarge the amount of data available for analysis by accessing information directly at its source. To integrate these into the classic border oriented system we create an abstract interface and management system, based on the Common Information Model.

Finally we will divide the management system itself into independent components, distribute them over the network and gain significant increase of performance.

1. Topical security systems

The interconnection of formerly private and isolated communication networks enables new forms of services and applications, but brings also new threats and the need for appropriate defense mechanisms. Until today, most networks are secured by firewalls which apply IP-packet-filtering at the interface between the internal and the external network.

This raises two major problems: First, as traffic is allowed or denied only based on IP-packet information, it is impossible to associate traffic to certain applications or process on the client machines in the internal network. If a client is infected by malicious software, collecting and sending information to an outside attacker e.g. through the standard HTTP port, the firewall may identify this as allowed traffic to a webpage server and hence allows the packets to leave the network.

On the other hand there are applications, especially for streaming media, which do not follow the classic client-server-model, where connections are always established by the client using well-known determined resources but use bidirectional peer-to-peer connections instead. They usually allocate these resources randomly, making it impossible to define an ip-based rule set without either understanding the protocol stream or gathering additional information on the client.

In some setups the problems stated above can be solved when analyzing the payload of the passing IP-packets. There are several different techniques available in the market, all of them basically do the same – they re-assemble the extracted payload to the originating higher-level protocol and apply filter rules to it. This method implies two things. First – the system must implement the specification of the watched protocol. In theory this problem can be solved, but in practice, many specifications are not available to the public and the number of different protocols raises every day. Additionally, as there is no standard for such filters, every system-vendor must build the filters for his own product. As soon as the payload is encrypted, there is no chance to gather information from the stream.

Due to these two problems, it is impossible to build a reliable and secure firewall system, when only inspecting traffic at the time it crosses the border. The only place where we can gather more data, is the source of the traffic, so we introduced a sensor on the endpoint of the network – the client-computer.

There are some commercial products available, that implement some of the techniques discussed in this article. All of them use proprietary control streams and interfaces so it is impossible to combine different products.

2. Client-side information gathering

The client-computer is the source of all traffic going through the network and crossing the border gateway. The most important information regarding our security system is the process that initiates the network packet. On a modern multi-task, multi-user operating system like Linux oder MS Windows a process has two significant properties. Each process is associated with a system user, who is the owner of the process and herewith creator and owner of the network packet. The second attribute we can assign to a process is the "program" that is executed by this process. To give an example – we can state that the connection to the webserver from "amazon.com" was triggered by the user "john" using the program located at "/usr/local/mozilla/mozilla-bin".

This information is very valuable for the security system. The program information tells us, that the requesting application is a legal webbrowser and not some malware, that tries to send out or download data. With the owner attribute we can of course simply check, if the user is allowed to use the resources, but we can do more – if a lot of process are started on different machines all over the network from the same user account, it is likely that the password of the user was compromised and someone is abusing this to gain access to restricted resources.

2.1. Technical prerequisites

All research within our group is based on the CIM Model [1] and uses the WBEM server from Sun Microsystems [2], which is written in Java. The decision for this WBEM implementation was taken, because we wanted a portable solution for at least the two operating systems that we are using in our group – Linux and Windows. Besides the portability the security aspect of the virtual machine concept was another reason for our decision.

To keep this advantages we wrote all other components in Java, too and tried to use the "Java Native Interface" Standard [3] for operating system dependent bindings.

2.2. Packet capture

As our approach centers on the network traffic, the packet is our starting point for all further analysis. So we begun with a component called "Packet Sensor Agent (PSA)". This program resides on the client computer and runs as a daemon in the background. Its task is capturing the network packets, preprocessing them and sending this information to a higher level application. The grabbing of packets unfortunately depends on the underlying operating system and is not available to the native Java language. The solution is the jpcap wrapper [4] – a java interface to the pcap packet capture library [5], which is available on the relevant systems. Both products are covered by open source licenses, so we can build a portable and royalty free packet capture program.

The jpcap creates a bundle of java objects for each captured packet, representing the different OSI layers of the packet. This internal format is optimized for speed when processing the captured data by the information of the single layers, but even if its open sourced, it is again a proprietary format, and not covered by standards. So the next step, was to find or create a structure in CIM, to put the data in.

CIM had no representation for network packets, but there were classes for network filters. Because filter and packet depend on each other, we decided to create a CIM structure describing packets based on the class "FilterEntryBase" and its subclasses. First we created a generic base class "PacketBase" as superclass for all network packet classes, then we derived the class "IPPacket" and "8021Packet". The attributes of the new classes were taken from the corresponding filter classes. Entries describing a range of values were converted to a single value entry, so "HdrSrcPortStart" and "HdrSrcPortEnd" from "IPHeadersFilter" became "HdrSrcPort" in "IPPacket".

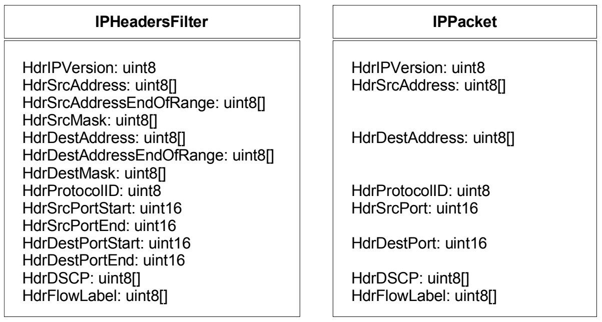

Figure 1. Packet and Filter for IP-Layer

Now we can represent the data in TCP/IP and UDP headers and the data of an ethernet frame. To stick ethernet and IP information on a packet together, we transfered the list/entry concept from the filter model. The "FilterList" merges multiple subclasses of "FilterEntryBase", we defined a "NetworkPacket" that can concatenate different packet classes to one packet.

2.3. Process information

The second interesting attribute is the process information related to a network packet. So who (user) and what (program) is responsible for the packet.

The CIM Model already has classes for process and user to model a whole operating system's state. It is possible to extract all wanted information from the model, but as our operating systems are not using such structures, their not available for our analysis. So we must find another way to gather this information, which is an operating system dependent task. As with the packet grabber we will use the java native interface to access the netstat program [6]. A tool that shows the relation between network resources, process and users.

In the next step we tried to put the gathered data into CIM classes. Using the existing schemes, the information needed for our approach is spread over a number of classes. This causes a huge overhead because we must create instances, not carrying useful data for us. So we finally decided to create our own structure "ProcessData" that simply carries the data for user and process and add it to the "NetworkPacket" in the same way as the single packets of the network layers. This implies, that " ProcessData" is a subclass of "PacketBase" otherwise we could not add it to "NetworkPacket". This is bit ugly, but as we do not need other classes at the moment this trick keeps things simple.

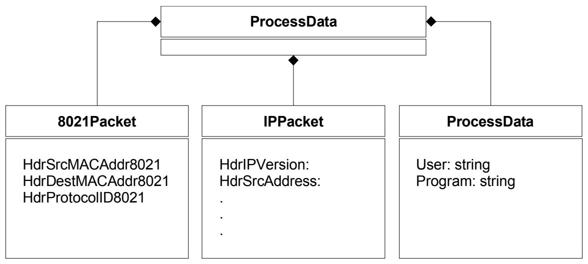

Figure 2. Class structure of a network packet

The better way would be, to derive one class each for user and process from a suitable hierarchy and make a branch with one list for each aspect like shown in this figure.

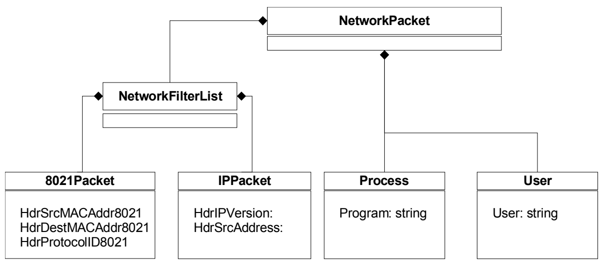

Figure 3. alternative structure for a network packet

2.4. Populate the information

Now it is time to populate the information to the border gateway. To do this in a convenient way, we will make the assumptions, that gateway and client share a common CIM repository.

When a packet is detected by the Packet Sensor Agent, the information is converted into CIM objects and put into the repository. The gateway can use time-based polling, to see if there are new packets available, and make configuration changes before the packet arrives at the border. Otherwise it can ask the WBEM server whenever a packet arrives at the border, to see if there is additional information available. The third method is, to inform the gateway when a new packet arrives at the repository.

2.5. Enhancements and problems

The system described in this chapter brings us one major feature - we can associate traffic with a process and user. This is useful to prevent certain kinds of malware from "phoning home" or abusing the network for sending viruses or opening backdoors. On the other hand it helps tracking legal software, that uses peer-to-peer communication and dynamic resource allocation. This recommends, that the gateway is able to understand the process information and includes it into the filter rules. Before taking a look at a possible implementation on the gateway, we have to discuss the problems associated with this approach.

One major problem is the performance of the system – in a busy network, the amount of collected packet information is enormous and will tear the central repository to the ground. Besides, the delay between detection and availability at the gateway is multiple times longer as the real packet needs to reach the gateway, so time critical applications, like videoconferencing, will not work well.

Another problem regarding availability is the central repository itself – as all information is collected and redistributed by it, a failure or breakdown of this system will bring down the whole network.

Under aspects of security, we must ensure that the agents collect and populate correct data and that it is impossible to fake or manipulate an agent. Both can enable an intruder to take over the total control on the network.

3. Traffic Management Provider

The Traffic Management Provider (TMP) is the missing piece between the capturing agent from the above chapter and the packet filter gateway at our network border. In general, a packet filter has a static rule set against which the passing packets are analyzed. No interaction with the outside happens for deciding about a packets way, so what we must do, is to plan and apply the packet filter rules based on packet data and the additional process information before the packet is processed by the filter.

The preferable method for the notification problem was the implementation of the TMP as a provider within the CIMOM. So we do not need to make another external connection with the server and can easily use the subscription mechanism to get notified on a new packet.

So the functional description for our TMP as a black box is like: receive information on new packets by subscribing to suitable CIM events, compare the available information against a policy database and establish an adequate packet filter rule on the gateway. The description consists of three separated tasks – the solution of the first one is mentioned above, the second is the most complex one and depends on the way we deal with the third - so we will start with the communication with the gateway.

3.1. Controlling a packet filter

When we talk about packet-filters we take the netfilter project [7] as a reference. The netfilter tools allow the filtering of IP and ethernet based traffic in a simple but effective way. The rules consist of a condition, which can test on nearly all parameters in ethernet and IP headers and additional on some parameters in transport layer protocols like ICMP, TCP and UDP, followed by one of the three actions drop, reject, accept.

Taking a look at the CIM scheme we can find a suitable description for the most items. The test condition will fit into the filter classes we already used as a blueprint to develop our packet classes. To build a rule, we can use the policy model . A policy in CIM is defined with a condition and an action part, so the policy condition side will carry a link to our packet filter classes and the policy action must contain simply a representation of one of the three possible actions.

We will represent the action part of the rule by an enumeration value in a class “PacketPolicyAction” derived from “PolicyAction”.

The representation of a complete filter entry is shown in figure 4.

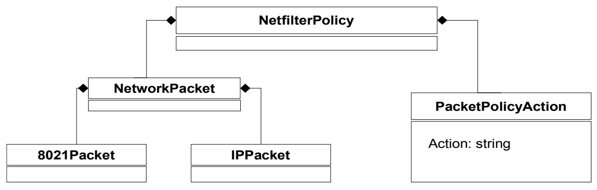

Figure 4. Policy describing a netfilter rule

To commit these PolicyRules to a real netfilter appliance we have three possibilities.

Building the appliance as a CIM client that polls data from the repository, registering a provider that subscribes on events related to the policy classes or connecting the appliance to the TMP via a proprietary network channel. Even if the first and second ones are the preferable way regarding the overall CIM approach, we chose a mixture of the first and the third one, due to performance issues. As we wanted a connection from the server to the client without the disadvantages of polling and all components in our system are written in Java, we decided to connect them via RMI and send the data as CIM instances. We can substitute this solution with a fully CIM standard one by establishing a CIM server on the netfilter client and writing the CIM instances onto this “local server”. The TMP provider will than act as client with the netfilter appliance.

3.2. Policy controller

The main task of the “Traffic Management Provider” is to decide if the incoming data-packet should pass the gateway or not. The decision is based on rules given by an administrator, these are held in the “policy lookup table” as a collection of instances of CIM's “Policy” classes.

In the preceding chapters we already talked about the policy model within CIM and how we modeled the network packet filters. To filter on the user and process information we receive from the client-side sensors we created “ProcessDataFilter” which simply contains two strings. Now we have everything we need to create a filter rule that we can compare against an incoming “NetworkPacket”.

When a new packet is detected by a client sensor, new instances of “NetworkPacket” and the associated classes are created on the CIM server. The creation triggers the event handler and notifies the TMP, where it is compared against the condition part of all policies in the “policy lookup table”. When a condition matches the packet, a new rule for the gateway is created. This is done by taking all parameters from the incoming packet, that are necessary for the rule to match and writing a new policy condition with these. The appropriate action for the netfilter gateway is taken from the rule and the policy is written to the netfilter client.

To filter on complex relations or enable content filtering the policy controller can be extended using a plugin interface.

4. Distributed services

Upon now we had a very simple setup with one security gateway and some clients that act only as a sensor. The concentration of all decisions to one point in the network makes the whole system slow and sluggish so we tried to distribute the single tasks over the network to decrease the necessary amount of requests to the central server and increase the level of security.

4.1. Active client

The first step for our improvement is the active client. Instead of only monitor and report data to the central server, the client gets its own local filter mechanism.

By default the client has no filter rules loaded into its rule base, so all traffic is denied and is unable to even enter the network. This prevents attacks inside the own network and decreases the useless traffic, that is send to the network and blocked on the border. The configuration of the clients filter rule set is done the same way like the netfilter gateway.

This implementation is good for security but increases the load on the central server, because it must now calculate and deploy the rules to the client. To get rid of this load, we must do the decision already on the client. We transfered some parts of the policy lookup table to the client and implemented a local policy controller, what enabled the client to make decisions on the fate of a packet without connecting the central repository. This additionally recommends, that packets which are checked and allowed by the client can cross the border gateway without further inspection. To ensure the consistency of the applied security rules we must keep track of the rules set locally and on the border gateway.

4.2. Network Security Service Manager

The "Network Security Service Manager" (NSSM) replaces the direct interaction of the TMP and the attached netfilter clients - as we will introduce security mechanisms other than netfilters, we will now start speaking of security devices that provide security services. So the task of the NSSM is to distribute the decision of the policy controller to the affected devices. Referring to the last chapter, we will make an example.

We have a PC in a company network and want to open a ftp connection to somewhere outside using an internal proxy server. The connection must pass three security devices within the company: first the PC's local firewall, the proxy server and the border gateway. The initiating connection is detected by the sensors of the client PC and reported to the TMP. There is a policy in the lookup table that allows the connection, and so the TMP decides to accept the packet. The initiating packet and the decision is now routed to the NSSM where we make a forecast on the upcoming related packets. Now the rule set is split into three parts, one for each of the security devices: For the proxy server and the border gateway we allow all packets that match the forecast. The local client receives the information about the forecast and is allowed to accept packets related to the originating request that are within the forecast.

This example implies two things, that have not been discussed until now. First, the NSSM must find all security devices that are involved processing the current request. As far as the request is attached to the network, we can do so by reading out the network structure from the CIM server. By extracting routing and topology information we can determine the way of each packet through the network and find the affected devices.

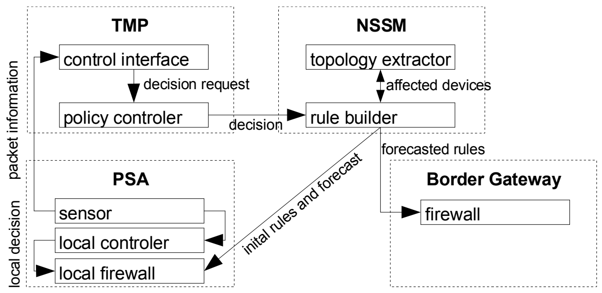

Figure 5: Distributed services

The second issue is not solvable with the CIM standard at the moment. To keep the filters effective, the NSSM must know about the capabilities of each security device, so we created a class tree to store information about a devices capabilities in the CIM.

5. Capabilities model

The capability model provides a set of classes that describe the features of a security device. The basic concept follows the already used policy model – we have classes describing the sensoric features of a device and such for the possible actions. Different instances are put together in a list and stored in the CIMOM with the instance of the security device itself.

The capability classes correspond with the existing filter classes, this means, the features addressed with an instance of "IPHeadersFilter" are collected in "IPHeadersFilterCapability" like shown in figure 6.

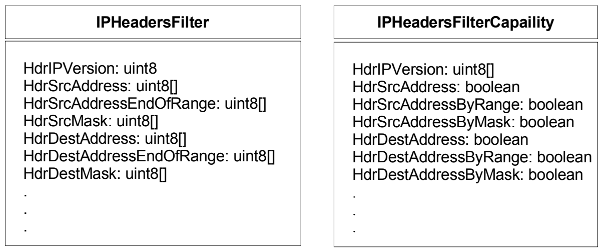

Figure 6: Capability class example

5.1. Class tree

To organize the capabilities in a homogeneous and continuous structure within the CIM we reworked the already existing structure of the network filter classes and based our hierarchy on this.

In accordance with the network filter scheme we decided to take the concept of multiple "Entries" that are concatenated in a "List". The starting points were labeled "FilterCapabilityEntryBase" and "Filter CapabilityList" respectively "ActionCapabilityEntry Base" and "ActionCapabilityList".

As we need filters and actions for different aspects of a computer system and not only for the network, we put these classes into the core-model. The above mentioned "IPHeadersFilterCapability" is a subclass of "FilterCapabilityEntryBase". Together with "8021FilterCapability" it can describe the features of a packet filter appliance. The class hierarchy diagram is shown in figure 7.

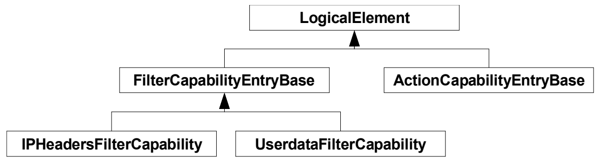
LogicalElement
FilterCapabilityEntryBase
ActionCapabilityEntryBase
IPHeadersFilterCapability
UserdataFilterCapability

Figure 7. Capability description classes

To keep the correspondence between capabilities and filter parameters, it is necessary to move the classes "FilterEntryBase" and "FilterList", which form the base for the network filters, up to the core-model.

These filter classes should not contain any action parameters, so we needed a similar structure for defining actions in the CIM standard and added "ActionEntryBase" and "ActionList" as counterpart of the filter base to the core-model.

5.2. Standardization

Most of the changes invented in this project were discussed during the weekly phone conference of the SPAM working group and have been improved with the feedback of the group. Currently we are reworking the first partial implementations and try to create a working prototype of the complete system. It is planned to discuss the model within the working group again and bring in the results into the standardization process.

6. Conclusion

We unfortunately could not implement all features we wanted to, because none of the components currently used supports CIM. So a lot of wrapper and interfaces were created, that have only a partial implementation of the originating features.

We think the presented work shows the necessity to introduce a capability description scheme and generalize and unify the filter and action scheme. In our opinion such a structure and its implementation in forthcoming products, would make it much easier to develop and deploy system wide management and security services in a network. Especially for heterogeneous networks with different types of operating systems or in public ones were no "single product" software solution is possible, the invention of standards will bring us closer to "plug & play security".

[1] CIM Schema v2.7, DMTF
http://www.dmtf.org/standards/cim/cim_schema_v27

[2] WBEM Services, Sun Microsystems Inc.
http://wbemservices.sourceforge.net

[3] Java Native Inferface, Sun Microsystems Inc.
http://java.sun.com/j2se/1.4.2/docs/guide/jni/

[4] jpcap – http://jpcap.sourceforge.net/

[5] pcap - http://www.tcpdump.org/

[6] netstat – network statisics tool

[7] netfilter http://www.netfilter.org/