IDS Administration Platform

MARCO AURÉLIO BONATO, WALTER GODOY JR.
Centro de Pós-Graduação em Engenharia Elétrica e Informática Industrial
Centro Federal de Educação Tecnológica do Paraná
Av. Sete de Setembro, 3165 Cep: 80.230-901 – Curitiba - Paraná
BRASIL

Abstract: - The goal of this paper is to present the existent open protocols for information exchange among components of an IDS (Intrusion Detection System)[1] architecture, how they work, and one open model, which is being developed by us to manage remote ids components. These protocols are considered open because they are not associated with any hardware and software manufacturer, but with autonomous groups, which look for standardization in which the cooperation among several products is possible, no matter the manufacturer.

Key-Words: - Intrusion Detection, Network Security, IDS.

1 Introduction

The more the attacks to computers grow the more the use of ids technology is justified, because Firewall based network protection can't detect attacks embedded in the application protocols or attacks that use networks, as DoS. In the case of large corporative networks, with many networks geographically distributed, protection based on Firewalls does not detect the attacks that occur inside the internal network[2].

There is one problem associated with the implementation and use of intrusion detection in large computer networks, which is the integration among the analyzers, because the analyzers of different manufacturers can't exchange information, and this becomes a problem for the security administrators.

To provide this portability among the components of an IDS architecture, a group of researchers of the IETF (Internet Engineering Task Force) has created a set of protocols called IDP (Intrusion Detection Protocol). This set of protocols can exchange information among IDS components. This work group is denominated IDWG (Intrusion Detection Working Group).

2 IDWG Specification

2.1 Intrusion Detection Message Exchange Format

The IDMEF[3] data model is an object oriented alert representation. With it it's possible to reach a pattern specification in the relationship between environments of little or great complexity.

The IDMEF data model guarantee:

- Heterogeneity: the alerts can be represented in many ways, depending on where they are installed and how the detection tool analyzes them. The model is flexible and because it is object oriented, it can represent these characteristics through the aggregation and subclasses functionality;
- Distinct environment: alerts of one given attack with different data sources, supply different information. The IDMEF data model defines classes that support different data for one given attack;
- Compatibility among analyzers: The model provides a set of format converters that will be used by the analyzers to supply the managers with standardized information. In order to define these extensions to the basic scheme, the IDMEF defines two kinds of alerts: the simple and the complex, both provided through the association and subclasses characteristics;
- Different analysis: depending on the environment in which the analyzer is inserted, attacks can be observed and reported differently. The specified model makes these differences flexible through subclasses which are defined with additional attributes, which can make the data sent to the management platform compatible.

2.2 Intrusion Detection Exchange Protocol

The IDXP [5] protocol is implemented as a profile of the BEEP[4] protocol, describing the way the information is exchanged among IDS components. While the BEEP model supplies the protocol, the IDXP specifies the necessary characteristics for the

establishment of a channel and information exchange among the involved components.

The IDXP specification can be divided in 4 parts, which are detailed below.

- Connections: for this task the IDXP profile requires the use of another profile called initialization. The use of the initialization profile for this purpose can preserve all the compatibility between the IDXP and the remaining phases, because they only need to know that there is a connection between two components and that it can be used;
- Security: after the establishment of the connection between two IDS components, it is necessary that the security in the information exchange also be established. As the initialization profile also provides the security characteristic, no other profile is needed for this functionality. All the security conditions are preserved at the moment when the information is transmitted;
- BEEP channel: after a BEEP session has already established and all the security processes have been initialized, the next phase is to open a channel where the data can be exchanged. The first message contains an URI, which determines how the information will be exchanged together, with a machine name an IP address of the origin. With this information the server has the capacity to decide if the client's request is acceptable, returning to it a yes or no. Through only one BEEP session, several IDXP channels can be created, this way saving the establishment from new sessions;
- Data transference: the data about intrusion detection is sent from the clients to the servers in one of the three types of MIME data: text/xml, text/plain or application/octet-stream. XML data must be in accordance with IDMEF messages. The other two types have been created in order not to restrict the use of new implementations.

2.3 Intrusion Alert Protocol

The IAP[6] protocol was constructed to carry alerts among the IDS components. Like any protocol, it is based on a model of communication and messages exchange. It is characterized by simplicity, not having the same robustness of BEEP/IDXP protocols.

3 Project Using Open Protocols

We are developing a project to manage the analyzers dispersed in a large network. Knowing that each IDS analyzer of a great corporative network will have different configurations, it will be necessary to remotely configure each one of them. And there's also the fact that not all the collected information is relevant to be sent to a centralized platform of networks management, therefore, through a configured filtering politics it is possible to specify which alerts will be received. As shown in Fig. 1, the Information Collector takes as a base for its functioning the rules that will be registered in the Configuration Manager. The Configuration Manager is a parameterized interface that the administrator uses to generate the profile of information collection for each local network that belongs to the corporative network and to specify the rules of functioning for each analyzer.

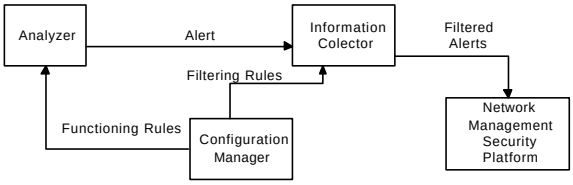

Fig. 1 Project Model

3.1 Configuration Manager

The Configuration Manager has the following characteristics:

- Responsible for the interface that makes the configuration of all the IDS analyzer available in the network;
- Responsible for the interface where the administrator can decide which alerts must be considered for each of the analyzers of the net;
- For the updating of the analyzer rules it is necessary a standard protocol that is shown below.

3.2 Intrusion Detection Analyzer Configuration Format

The Intrusion Detection Analyzer Configuration Format has as its function to make possible to IDS analyzers to be configured remotely. The relationship between the principal components of the data model is shown in Fig. 2.

To construct this model the model IDMEF[3] was used as a base.

The top-level class for the model is IDACF; the configuration is a subclass of this top-level class.

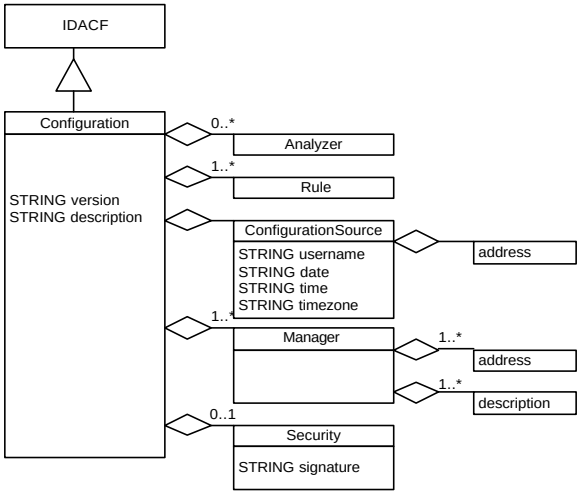

Fig. 2 IDACF Model Representation

The Configuration class has two attributes:

- version: required. The version number of the rules. The creation of the rules is automatic and has the following format: YYYMMDDSS. To update the rules version is necessary to load the old configuration, so the system can generate the new one. YYYY – year, MM – month, DD – day and SS – sequential number that receives 01 when any modification occurs in the year, month or day.
- description: optional. One description for the configuration rules. Example: "rule for DMZ analyzers".

3.2.1 The Analyzer Class

The Analyzer class Fig. 3 identifies the analyzer that receives the functioning rules. For belonging to an environment where more than an analyzer can have equal configurations, the model permit that a group of rules can be attributed to more than an analyzer.

The Analyzer class and its aggregations are defined in [3].

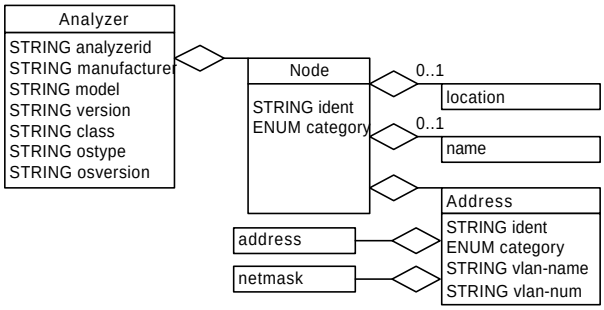

Fig. 3 The Analyzer Class

3.2.2 The Rule Class

The Rule class is used to inform to the analyzer what are its rules of functioning.

The Rule class is composed of seven aggregated classes, as shown in Fig. 4.

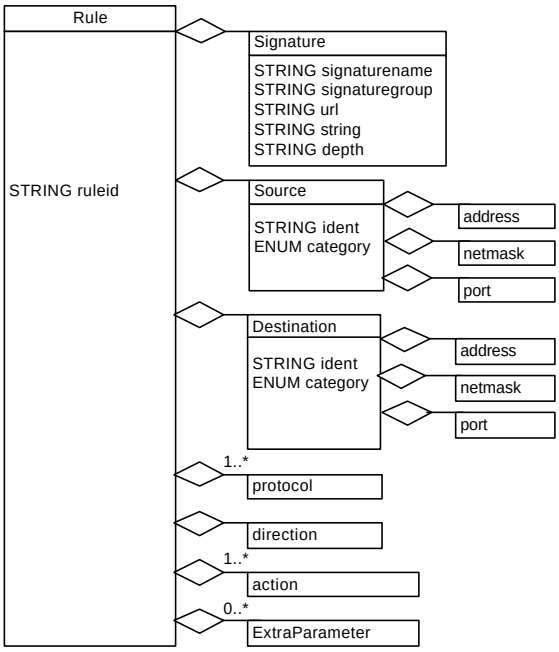

Fig. 4 The Rule Class

The Rule class has three simple aggregated classes that are defined here:

- protocol: one or more. STRING. The protocol that will be analyzed. The permitted values for this class are defined in the RFC790[7]. If it is not possible to define the protocol, the default value is "any";
- direction: exactly one. STRING. Define the search criteria. To search for packets going to or originating from a network machine. The permitted values for this class are shown below. The default value is "s<>d".

Rank	Keyword	Description
1	s->d	From the origin to the Destination
2	s<-d	From the destination to the Origin
3	s<>d	To the two directions

Table 1 The direction definition

- action: one or more. STRING. The action that will be executed by the analyzer when the conditions defined in this rule occur. The permitted values for this class are shown below. The default value is "log".

Rank	Keyword	Description
1	log	Save the information in the log
2	Alert	Save the information in the log and send an alert to the security network management
3	reset_sender	Send information to the sender to close the connection.
4	reset_destination	Send information to the destination to close the connection
5	reset_all	Send information to the sender and destination to close the connection

Table 2 The action definition

The Rule class has one attribute:

- RuleID*:* optional. A unique identifier for the rule;

3.2.2.1 The Signature Class

The Signature class is used to define the signature characteristics.

The Signature class has five attributes:

- name: required. The signature name. This name will be stored in the log or showed to the administrator when the rule is detected by the analyzer;
- group: optional. The group to which the signature belongs. This attribute is used to group signatures that have characteristics in common. Example: ftp (signatures associates to FTP protocol), http (signatures associates to http protocol), etc;
- url: optional. A url at which the manager can find additional information about the signature;
- string: required. The string definition. This is the content that the analyzer will search in the packet;
- depth: optional. This sets the maximum search depth for the content pattern match function to search from the beginning of its search region.

3.2.2.2 The Source Class

The Source class is used to describe the source machine that will be analyzed.

The aggregate classes that make up Source are:

- address: zero or one. STRING. The source address information. The format of this data is governed by the category attribute. If it is not possible to define the source address, the default value is “any”;
- netmask: zero or one. STRING. The network mask for the address, if appropriate;
- port: zero or one. STRING. The source port information. This class can be used for protocols Udp and Tcp. If it is not be possible to define the source port, the default value is “any”.

The Source class has two attributes:

- category: required. The type of address represented. The permitted values for this attribute are shown in [3]. The default value is “any”.

Rank	Keyword	Description
15	Any	Any address

Table 3 New category for the address class

- negate: required. Address negation. This attribute allow the administrator to invert the meaning of the action. The permitted values for this attribute are “yes” or “no”. This is useful when the administrator needs to create one rule where only one source address can bypass the rule.

3.2.2.3 The Destination Class

The Destination class is used to describe the destination machine that will be analyzed.

The aggregate classes that make up Destination are:

- address: zero or one. STRING. The destination address information. The format of this data is governed by the category attribute. If it is not possible to define the destination address, the default value is “any”;
- netmask: zero or one. STRING. The network mask for the address, if appropriate;
- port: zero or one. STRING. The destination port information. This class can be used for protocols Udp and Tcp. If it is not possible to define the destination port, the default value is “any”.

The Destination class has two attributes:

- category: required. The type of address represented. The permitted values for this attribute are shown in [3]. The default value

is “any”;

- negate: required. Address negation. This attribute allow the administrator to invert the meaning of the action. The permitted values for this attribute are “yes” or “no”. This is useful when the administrator needs to create one rule where only one destination address can bypass the rule.

3.2.2.4 The ExtraParameter Class

The ExtraParameter class is used to define additional parameters that will be sent to the analyzer. These parameters are defined by the analyzer’s features implementation, and can be different for each analyzer.

The ExtraParameter class has two attributes:

- parameter: optional. The name parameter that will be informed to the analyzer;
- value: required. This attribute is required if the attribute parameter is informed. It is the value of the parameter.

3.2.3 The ConfigurationSource Class

The ConfigurationSource class is used to inform which machine sent the configuration rules for the analyzer.

The aggregate class contained in ConfigurationSource is:

- node: exactly one. Information about the host or device that sent the configuration to the analyzer.

The ConfigurationSource has tree attributes:

- username: required. The user that sent the configuration rules to the analyzer;
- date: required. The date when the rules were sent to the analyzer. The data has the following format: YYYYDDMM;
- time: required. The time when the rules were sent to the analyzer. The time has the following format: HH:MM:SS.
- timezone: required. The time zone.

3.2.4 The Manager Class

Knowing that on a large network can exist more than one platform of security management, the Manager class identifies which are these platforms and the analyzer knows who sent the alert messages.

The aggregate class contained in Manager is:

- node: one or more. Information about the host or device that is the Network Security Manager.

3.2.5 The Security Class

The security between the Configuration Manager and the analyzer is achieved by the Beep protocol characteristics. Additionally to this security, the analyzer receives, through the Security class, one digital signature that can be used by the analyzer to prove the rules consistency.

The Security class has one attribute:

signature: optional. The digital signature that is sent to the analyzer.

3.3 The IDACF Tests

The tests related below (Fig. 5) were done to show if the IDACF can be useful. To perform these tests two APIs for Linux were created to permit that a Snort analyzer could be configured by a Web Server.

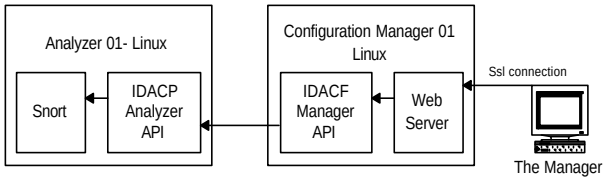

Fig. 5 The IDACF Tests

The protocol was successfully tested and now we are testing it in a commercial IDS analyzer.

3.3.1 The Next Steps

After the tests, to complete the platform describe in Fig. 1 we need:

1) To improve the functioning of the Configuration Manager. The strength of this platform is in its facility to configure the analyzers. During the tests we detected that the configuration manager must have specific features for different analyzers. The problem now is not the way that the rules are sent to the analyzer; it is to find the best way to implement the Configuration Manager;
2) Implement Beep/Idxp in the data exchange between the Configuration Manager and the analyzers;
3) Implement IDMEF in the data exchange between analyzer and the Information Collector;
4) Define the functioning rules for the Information Collector to do the activity as shown in Fig. 6;
5) Test the platform and comment the results.

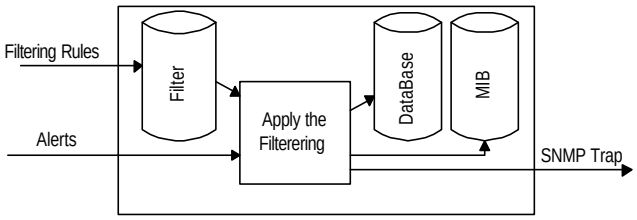

Fig. 6 Information Collector

5 Conclusions

It's getting more and more complicated to manage security in great corporate networks, because the environment is extremely complex and there's a great variety of hardware and software.

This work showed which are the basic requirements for information exchange among components of an IDS architecture where the focus is a centralized administration. Inside the same network it is possible to have IDS platforms of different manufacturers exchanging information. The platform that is being proposed here is fundamental when you can manage the security of remote networks.

References

[1] Biswanath Mukherjee and Karl Levitt, Network *Intrusion Detection*, IEEE Network, 1999.

[2] Rebeca Bace and Peter Mell, *NIST Special Publication on Intrusion Detection Systems*, 2000.

[3] D. Curry and H. Debar, *Intrusion Detection Message Exchange Format data model and Extensible Markup Language (XML) Document Type Definition. February*, 2002.

[4] M. Rose, *RFC3080:The Blocks Extensible Exchange Protocol*, March 2001.

[5] G. Mattnews and et all, *The Intrusion Detection Exchange Protocol (IDXP). Internet Engineering Task Force*, 2002.

[6] D. Gupta and et all, *IAP: Intrusion Detection Protocol, Internet Engineering Task Force*, 2001.

[7] Postel J., *RFC0790: Assigned Numbers*, 1981.