A Remote IDS based on Multi-agent Systems, Web Services and MDA

Mauro Silva, Denivaldo Lopes and Zair Abdelouahab
Department of Electrical Engineering
Federal University of Maranhão, UFMA
São Luis, Brazil
maurocsilva@gmail.com, dlopes@dee.ufma.br and zair@dee.ufma.br

Abstract— In this paper, we present a model, an architecture and an implementation of a Remote IDS (Intrusion Detection System) using the technology of Multi-agent Systems, Web Services and MDA (Model-Driven Architecture). This model adapts and extends the NIDIA (Network Intrusion Detection System based on Intelligent Agents) to provide a remote IDS on the Internet. The purpose is that users that do not have a local IDS can use the services provided by our remote IDS. NIDIA is an IDS whose architecture consists of a set of cooperative agents. The IDS functionalities are provided as a set of accessible services on the Internet through Web Services. The architecture of our IDS uses MDA to support metadata management such as profiles of machines, profiles of users and profiles of services. An illustrative example shows our IDS.

Keywords-component; intrusion detection; Software security, privacy, agents, metadata repository, Model-Driven Architecture

I. INTRODUCTION

Several techniques and tools are available to assist in the security process, such as Firewall, Anti-Virus and IDS (Intrusion Detection System) [1]. According to Pikoulas [1], the objective of the IDS is to identify directed threats to an organization and, later, to guarantee that the system is protected.

IDS systems are classified as centralized or distributed [1]. A Centralized IDS is the one in which data analysis is carried out in one host, independently of the number of hosts that are monitored. A Distributed IDS is the one in which data analysis is carried out in several hosts.

A centralized IDS has advantages such as simplicity and direct access to data and disadvantages such as bottleneck (all processing is concentrated on one machine). Some distributed IDSs have been developed, such as DIDS [1], AAFID [2], and the NIDIA [3]. These IDSs are designed with the objective to monitor hosts and network packets.

In this paper, we propose the adaptation and the extension of the architecture of IDS NIDIA [3] to provide remote IDS services on the Internet. This allows users who do not have an IDS make use the IDS services provided by NIDIA on the Internet. For this purpose, we have created an IDS model that is implemented using multi-agent, Web Services and MDA technology.

The organization of this paper is presented as follows. In section 2, we give one brief introduction on the multi-agent systems, Web Services and MDA. In section 3, we describe the model and the architecture of IDS NIDIA. In section 4, we present the adaptation and the extension of NIDIA to provide their services on the Internet. In section 5, we discuss the advantages and disadvantages of the new NIDA and present some directions of future research.

II. BACKGROUND

The new NIDIA model is being developed with Multi-agents Systems, Web Services and MDA. Each technology is used to play a well defined functionality. Multi-agent systems provide detection, analysis and reaction to threats. Web Service is necessary to allow services to be accessed by local and remote users. MDA provides a model driven approach to handle the NIDIA profiles, i.e. metadata and data.

A. IDS and Multi-agent Systems

A multi-agent system consists of a collection of agents that can interact together to learn or to change experiences. Agents are flexible because they have properties such as reactivity, pro-activeness and social ability. The reactivity means that the agent perceives the environment and provides answers to satisfy its design objective. Pro-activeness means that the agent is able to exhibit goal-directed behavior by taking the initiative in order to satisfy its design objectives. The social ability means that it can interact with other agents to satisfy its design objectives.

Some projects have used the technology of agents in detection systems of intruders. Amongst these, we can cite AAFID[2], JAM[4], and SPARTA[5].

AAFID Project has its development based on Perl agents. The communication between the agents does not have a definite flow. The agents communicate with all others. This project is based on analytical agents. JAM Project uses intelligent and distributed agents in Java and data mining to learn about intrusion models and behaviors that can be shared between organizations. The SPARTA uses the paradigm of...
mobile agents in the execution of tasks that go since the management of networks until the intrusion detection.

B. Web services

Web services [7] are designed to support interoperable machine-to-machine interaction over a network. Web services are self-contained, self-describing, modular applications that can be published, located and invoked across the Web.

There are three main components in a Web service environment: SOAP (Simple Object Access Protocol) [7], WSDL (Web Services Description Language) [7] and UDDI (Universal Description, Discovery, and Integration) [12]. These components are based on XML (eXtensible Markup Language).

SOAP is the standard protocol for accessing web services, making possible the communication between applications and its information interchange. It is an XML-based protocol for messaging and remote procedure calls which works on existing transport protocols, such as TCP, HTTP, SMTP, etc. A SOAP message is an XML document with a structure consisting of four basic parts: envelope, header, body and fault.

WSDL is an XML-based language for describing Web services and how to access them. It specifies the localization of the service and the operations that it requires. It shows a public interface of a Web service.

Finally, UDDI is a registry where available services are published. Thanks to the registry, end users can easily discover Web services information.

We can now define a Web service in terms of these technologies as: An XML Web service is a software service exposed on the Web through SOAP, described with a WSDL and registered in UDDI.

C. MDA

MDA [8] (Model-Driven Architecture) is a system development approach driven by models, i.e., models are used in all software development phases. There are three primary goals of MDA: (1) portability, (2) interoperability and (3) reusability. It addresses architectural separation of concerns. To achieve the above goals, MDA proposes several specifications and tools for using models in software development, e.g. MOF (Meta-Object Facility) for metamodeling, UML for modeling, XMI (XML Metadata Interchange) for exchange models and metamodels among tools, and model transformation languages.

MOF [9] defines an abstract language and framework for specifying, constructing, and managing technology-neutral metamodels. In fact, MOF defines a framework for implementing repositories that hold the metadata described by a metamodel. XMI is a XML format for meta-data and data interchange among repositories.

MDR (Metadata Repository) [10] is a tool that allows the manipulation and persistent state for metamodels created using MOF. As its name suggests, the MDR is a repository which implements the MOF. Thus, MDR is qualified to load, manipulate and store metamodels compliant to MOF and their corresponding models. Metamodels can be imported to MDR or be exported from MDR using XMI standard. Repository MDR can be managed through routines using APIs compliant to JMI (Java Metadata Interface) [11].

III. IDS NIDIA

IDS NIDIA is an IDS based on agents to detect and to deal with security incidents from data collected from logs of hosts, and packets of network traffic and generation of indexes of attack suspicion, and then to take countermeasures [3]. Fig. 1 illustrates the IDS NIDIA that monitors an administrative domain composed of some local networks and a collection of Sensor Agents. This first version of NIDIA can only provide IDS functionalities for hosts in a local network. In section 4 and section 5, we present some adaptation and extensions for the NIDIA in order to provide IDS functionalities in the Internet for external users.

Figure 1. Monitoring Strategy of NIDIA [3]

A. NIDIA Architecture

Fig. 2 presents the NIDIA model that is structured in layers [3]. Each layer has a set of agents.

![Layers of the NIDIA](image)

1) Monitoring Layer: is responsible for catching the occurrence of events and passing them to other layers. A class of SMA Agents (System Monitoring Agent) is located in this layer. It has the following agents [3]:
a) **Network Sensor Agents:** responsible for capturing the packages that are transported through the network.

b) **Host Sensor Agents:** they work collecting information in real time in one host and pass this information to the analysis layer.

2) **Analysis Layer:** is responsible for the analysis of the events received from the monitoring layer. It formats the collected events to identify attack patterns and it can check if an event is a true attack or not. In this layer, SEA agents (Security Evaluation Agent) are located. The Agent SEA makes use of a neural network. The neural network is responsible for helping in the recognition of attack standards. The BackPropagation algorithm is used in the learning of the neural network.

3) **Reaction Layer:** is responsible for taking countermeasures when a security problem is detected. To take decisions, the databases of strategy and actions is used. In this layer, SCA agents (System Controller Agent) are located.

4) **Update Layer:** is responsible for update the information in the databases. The consultations can be made directly from any layer; however insertions can only be made through this layer. It has the responsibility to keep the integrity and consistency of the stored information. In this layer, SUA agents (System Updating Agent) are located.

5) **Administration Layer:** is responsible for the administration and integrity of all agents of the system. In this layer, MCA agents (Main Controller Agent) are located.

6) **Storage Layer:** is responsible for managing the persistent information in IDS-NIDIA. In this layer, the databases of NIDIA are located.

In the next sections, we present the adaptation and extensions for NIDIA needed to provide IDS services on the Internet.

### IV. ADAPTATION AND EXTENSION FOR NIDIA

New requirements and concepts were introduced in the NIDIA model and architecture in order to adapt and extend NIDIA to provide IDS services on the Internet and to a distributed IDS.

#### A. New requirements and solutions for the NIDIA

The new NIDIA system must be able to manipulate a large set of information, protect many users and detect many attacks. Once an attack is detected by NIDIA, the used countermeasures are stored in a database for future reutilization.

The new requirements are listed as follows:

- **NIDIA Profiles** has the information about the available remote services in NIDIA. A NIDIA profile describes the services that an external user can request. The profiles can assist in the integration of NIDIA to other tools (Firewalls, Anti-Virus), allowing a user to request the use of the IDS integrated to a Firewall or to an Anti-Virus.

- **Remote Management Layer** coordinates the NIDIA agents that can be located in different networks. This layer enables NIDIA agents to be also distributed on the Internet.

- **IDS-Client** is a component used by the user to request the services provided by NIDIA on the Internet.

- **IDS-Proxy** intermediates the interactions between the IDS-Client and the Remote Management Layer. IDS-Proxy allows the IDS-NIDIA to provide their IDS functionalities as services through Web Services. Thus, allowing an external user to request the remote IDS.

#### B. NIDIA Profile

The new NIDIA is based on Web Services and MDA. On the one hand, the interoperability between the agents and other components of NIDIA is realized thanks to Web Services. Models, metamodels and transformation among models using MDA concepts and tools enable the data and metadata management. The metamodels created for NIDIA are conforming to MOF. These metamodels are created to enable the modeling of profiles.

This metamodel is stored in the MetaData Repository (MDR). MDR allows the manipulation of the metamodel elements. This repository allows the creation, initialization and exclusion of metamodel element instances, i.e. model elements. For this purpose, a library in Java and an interface according to Java Metadata Interface (JMI) is developed. This library and the interface provide also the functionality of exporting profile models in the XMI format. This profile model in the XMI format is transmitted to or shared with other agents or components of NIDIA. Thus, our approach for handling profiles as models provides some benefits such as interoperability, mobility and uniformity of data. For example, an external user knows the functionalities of NIDIA just by accessing the profile models. A profile model can also be used to configure and control the NIDIA system.

#### C. New concepts for NIDIA

In order to provide the IDS services on the Internet, new concepts must be introduced in NIDIA. These concepts are introduced using agents as follows:

- **StartNIDIA** is responsible for initiating the application, creating an agent LSIA (Local Security Intelligent Agent) in the main-container and starts the agent RMA (Remote Monitoring Agent).

- **LSIA** (Local Security Intelligent Agent) initiates the agents of NIDIA through a configuration file “agents.txt”. In this way, the hierarchy of wrapper agents containing Host agents is created. It is also responsible for managing the metadata repository that is used to manipulate the information on the agents.

- **NSA** (Network Sensor Agent) initiates the network sensor. In this class, two behaviors are basic: CapturePackage and SendPackage. CapturePackage initiates the capture of packets in the network, and the SendPackage gets the captured packets and sends them...
to SMA agent. For this purpose, we use the library Java Package for Packet Capture (JPCAP) [14].

- **SMA** (System Monitoring Agent) formats the data that were collected by the NSA.

- **Wrapper Agent** intermediates the communication among other agents of NIDIA. Each NIDIA agent (except a wrapper agent) has a corresponding agent of this type. It receives the messages and distributes them to other agents that are less loaded. Thus, a wrapper agent provides one better use of the system.

- **SEA** (Security Evaluation Agent) receives messages from the agent SMA, proceeds with the message analysis, and sends to MCA a message containing the identification of the connection and the degree of attack severity.

- **MCA** (Main Controller Agent) makes the control and monitoring of the activities of the other agents. It receives information from the SEA agent, in order to initiate a reaction. For this purpose, it requests the agent BAM to provide a reaction strategy.

- **BAM** (Binary Association Memory) realizes the identification of the type of attack. It receives from the MCA agent a binary vector of data and identifies the type of attacks. It uses a neural network for identifying the attack type and returns it to the MCA.

- **SSA** (System Sentinel Agent) is responsible for detecting if an agent is alive or not. For this purpose, it sends ping messages to other agents and it waits the return message. If an agent does not respond before an elapsed timeout, then the SSA notifies the Remote Management Layer that realizes the execution of a fault tolerance strategy.

**D. Remote Management Layer**

The remote management layer is responsible for the coordination of agents which execute remote services. Agents that belong to this layer realize the following activities:

1) They read the Profile filled by the user, and indicate which Agents are necessary to execute the requested tasks.
2) They select the agents that must be migrated.
3) They intermediate the communication among other agents.

This layer is composed of six agents: remote monitoring agent, remote analysis agent, remote reaction agent, remote update agent, remote administration agent and remote storage agent. Each agent in the management layer manages a suitable layer of NIDIA. Fig. 3 presents the new NIDIA architecture.

According to Figure 3, we can deduce the existence of a significant communication overhead between the Agents of Remote Management layer and the agents of other layers. In order to minimize communication overhead, a set of Agents were designed to play the role of facilitators and communication manager between the layers.

**E. IDS-Client**

IDS-Client allows external users to access remote IDS NIDIA. IDS-Client has a direct relationship with the IDS-Proxy. IDS-Client is a customer application that is used by an external user to find, choose and access IDS remote services. Fig. 4 presents the fundamental elements of IDS-Client.

Once IDS-Client is installed in the user machine, the system diagnostic acquires machine information, such as disk space, free memory, and idle time of the processor. This information is used to choose facilities to be installed in the user machine. Facilities are additional elements for the IDS-Client that provides functionalities such as a basic or graphical user interface, network packet capture, user notification and countermeasures. After this installation, IDS-Client calls up the IDS-Proxy in order to the later creates a remote IDS environment that provides IDS services. Once the configuration is ready, the IDS-Client can start the search for threats.
V. COMMUNICATION MODEL FOR IDS-NIDIA

Fig. 5 presents the communication model for IDS-NIDIA. This communication model can be detailed according to the following steps:

- IDS-Client requests a connection with the IDS-Proxy (IDS-Client sends an OpenConnection message to IDS-Proxy).
- IDS-Proxy accepts or refuses the connection (IDS-Proxy returns a ConnectionResult to IDS-Client).
- If IDS-Proxy accepts the connection, then IDS-Client uses the System Diagnostic component to obtain the information of user machine. Then, IDS-Client sends the information machine (IDS-Client sends the message SendDataMachine).
- IDS-Proxy analyses the machine information, and selects and returns other components to be used in the IDS-Client (IDS-Proxy returns the message FacilityComponents that contains the other components).
- IDS-Client requests the NIDIAProfiles (IDS-Client sends the message GetNIDIAProfiles). Then, IDS-Proxy calls up the RemoteManagement that, in its turn, requests to NIDIA-Agents their capabilities. Once RemoteManagement receives the capabilities, it creates a NIDIA Profile and returns it to the IDS-Proxy (IDS-Proxy sends the message NIDIAProfile).
- IDS-Proxy receives the NIDIA-Profile and sends it to the IDS-Client (IDS-Proxy sends the message NIDIAProfile).
- IDS-Client presents to the user the IDS services, and then the user selects the IDS services. Afterwards, IDS-Client requests the selected IDS services. IDS-Proxy requests to RemoteManagement to create an IDS Remote Environment. Then, IDS-Proxy returns the IDS services allocated to be used by the IDS-Client.
- IDS-Client captures data such as network packages, host logs, and so on. Then, the client calls the IDS services passing as parameters the captured data (IDS-Client sends the message Call-IDS-Services). The IDS Remote Environment executes the requested services, and returns the analyses results (IDS-Proxy returns the message IDS-ServiceResult).
- IDS-Client takes the countermeasures defined by the IDS Remote Environment (IDS-Client sends the message TakeCounterMeasures to itself). For this purpose, IDS-Client can use the local countermeasures components or request the support of the IDS Remote Environment. For example, in the case of a DoS (Deny of Service) attack, the IDS-Client can only use the local countermeasure component to close the connections with the malicious host. In the case of Worm attack, the local countermeasure component can request the support of the Remote IDS Environment to provide a honeypot strategy to face this kind of worm.

Once IDS-Client finishes the countermeasures, the IDS-Client presents the results to the user.

VI. IMPLEMENTATION AND TESTS

Our prototype of Remote IDS has been developed using the JADE agent platform, Sun’s Java Development Kit 1.4.2_05, MDR Repository for NetBeans 3.5.1 and Systinet for Java 6.5.

The new IDS NIDIA (IDS Proxy, Remote Manager and Agents NIDIA) and the IDS Client have its main functionalities implemented, and some tests have been realized to test our prototype. Fig. 6 presents the scenario that is used for the tests.
A user that does not have a local IDS can call up the IDS-NIDIA through the IDS-Proxy. The former is a component for the user, and the latter is a component for the NIDIA. We have simulated several attacks in the User Domain such as DoS (Denial of Service) and Worms. Our prototype has detected these threats and provided countermeasures such as disabling the connection with the attack source (malicious host) or redirecting the connection to a honeypot.

VII. CONCLUSION

In this paper, a remote IDS model is proposed. This model is designed in order to provide IDS services on the Internet. A prototype based on the remote IDS model is implemented in JADE platform, using Web Services to provide interoperability, and MDA to manage profile models.

In this way, users that do not have local IDS can make use of the IDS services on the Internet provided by our prototype. Some tests have been realized to evaluate our prototype. The partial results have demonstrated the advantages and disadvantages of our prototype.

The advantages of our proposed tool are: IDS is provided as Web services, then it can be easily used by external users; IDS-NIDIA is based on intelligent agents that use neural networks, then it can learn and use the learned knowledge to take efficient countermeasures; IDS-NIDIA is based on Web Services and MDA, then it can be easily integrated with other tools; the IDS functionalities can be distributed on a local network or in remote networks, balancing the load processing.

The disadvantages of our proposed tool are: a large communication flow between the IDS-Proxy and Remote Management; IDS-Proxy and Remote Management are not distributed between different domains (local networks) unlike the other NIDIA layers that can be distributed among several networks.

In the future, we aim to adapt NIDIA to other tools, such as FireWalls and Anti-Virus in order to improve our remote IDS to make face to a wide variety of threats. As well as a study on the scalability of the model, the Web Services and the Agents.

REFERENCES

[9] OMG, MOF 2.0 Versioning and Development Lifecycle Submission, OMG document ad/03-02-08.