Intusion Detection Prototype Based on ADM-Logic

Mehdi Talbi, Meriam Ben Ghorbel-Talbi
Digital Security Unit, Higher School of Communication, SUP'COM Tunis, Tunisia
Email: mehdi.talbi@rennes.supelec.fr, meriam.benghorbel@enst-bretagne.fr
Mohamed Mejri
LSFM Research Group, Computer Science Department, Laval University, Quebec, Canada
Email: momej@ift.ulaval.ca

Abstract— **Intrusion detection systems (IDS) are considered nowadays as one of the most important components in the security architecture of information systems. For a Misuse-based IDS, also known as signature based IDS, the efficiency of detection is highly correlated to the quality of signatures. It is therefore very important to select a suitable formal language that provides both high expressiveness and simplicity when specifying attack signatures. It is also fundamental to have a user friendly and automatic tool allowing the specification and the verification of these signatures. This paper shows the efficiency and the suitability of the ADM-logic as a formal language to specify properties characterizing a large variety of attack scenario, and focus on the design and implementation details of our intrusion detection prototype based on this logic.**

Index Terms— **intrusion detection system, ADM-Logic, TCP-IP based attacks**

I. Introduction

Considering the increasing number of attacked systems and their financial effects, information systems must be henceforth protected against intrusions attempts. This protection is provided by several tools including Intrusion Detection Systems : IDS. These detection systems are divided into two families : anomaly-based detection and misuse-based detection. Regarding the second approach, protection is usually provided by parsing the traffic or audit files in order to detect matches with patterns that are previously defined in what we call database of attack signatures. It is therefore very important to have a formal and expressive language to specify these signatures.

The main purpose of this paper is to propose the use of the ADM-Logic [1] as a formal language for misuse-based IDS. A wide variety of TCP/IP attacks will be specified using this logic showing its suitability and its expressiveness. Finally, a prototype of an IDS, called GTM-System, using this logic will be detailed.

The remainder of this paper is organized as follows : Section 2 introduces the ADM-logic as a signature specification language, some TCP/IP attacks and their specification using this language are given in Section 3, while Section 4 presents the GTM-System. Section 5 discusses related work, and finally, some concluding remarks on this work and future research are ultimately sketched as a conclusion in Section 6.

II. Signature Specification Language

The IDS proposed in this paper uses the classical model checking technique to verify whether or not a model satisfies a formula, i.e. $\mathcal{M} \models \phi$, where $\mathcal{M}$ is a model and ϕ is a formula. Intuitively, the model is an abstraction of events in some audit files and the formula is specified in the ADM-Logic. The satisfaction relation $\models$ can be seen as a function that takes a model and a formula and returns true or false. In the case of false, some additional information could also be returned to help the user finding the reasons behind the failure.

Hereafter, we formally define the model together with the syntax and the semantics of the ADM-logic.

A. Model

One of the basic steps of intrusion detection is the audit trail analysis. It allows to record and analyze some particular actions that have been performed on a system during a given period of time. An example of these analyses is to make detection of particular sequence of events characterizing an attack signature.

Following the approach presented in [2], we use a linear based-trace model, where trace is defined as a sequence of events (actions) collected from an audit source. The set of traces, denoted by $\mathcal{T}$, is defined as follows :

$$\begin{cases} \epsilon \in \mathcal{T} \\ \text{if } \tau \in \mathcal{T} \text{ and a is an action, then } \tau.a \in \mathcal{T} \end{cases}$$

Where ϵ stands for the empty trace and "." is the concatenation operator in sequences.

Since most of network-based attacks are due to TCP/IP related protocols we define internet actions as an abstract notation of packet represented by a sequence of fields denoted as following :

- TCP actions : (tcp.ipsrc.ipdest.portsrc.portdest.flags.frags)
- UDP actions : (udp.ipsrc.ipdest.portscr.portdest)
- ICMP actions : (icmp.ipsrc.ipdest.type.code.frags)
- ARP actions : (arp.ethsrc.ethdest.opcode.macsrc.macdest)

Where :

- ipsrc, ipdest = $a_1.a_2.a_3.a_4$, with $a_i \in \{0, \ldots, 255\}$. (Source and Destination IP addresses).
- ethsrc, ethdest = $e_1.e_2.e_3.e_4.e_5.e_6$, with $e_i \in \{\backslash x00, \ldots, \backslash xFF\}$. (Source and Destination MAC addresses in the Ethernet header).

- macsrc, macdest = $m_1.m_2.m_3.m_4.m_5.m_6$, with $m_i \in \{\backslash x00, \ldots, \backslash xFF\}$. (Source and Destination MAC addresses in the ARP header).
- flags = urg.ack.psh.rst.fin.syn (TCP flags).
- frags = fo.mf (fragment offset and more fragment fields).

B. ADM-Logic

The ADM-Logic, proposed in [1], can be viewed as a special variant of μ-calculus [3] that requires a linear model (trace-based model). Initially designed for the specification of electronic commerce properties, it is also very appropriate for the intrusion detection issue. Compared to the Linear Temporal Logic (LTL) [4], the ADM-logic provides more expressiveness. For example, the ADM-logic allows the specification of properties that require counting (e.g. number of action "a" is equal to the number of action "b"), which is not possible with the LTL or the μ-calculus. The logic has a denotational semantics and it is endowed with a tableau-based proof system that leads to a modular denotational semantics and local model checking which is useful for an efficient implementation.

Syntax. The syntax of this logic is based on patterns that are sequences of actions and pattern variables as shown by the following BNF-grammar :

$$p := a.p \mid v.p \mid \epsilon$$

where ϵ stands for empty pattern, a is an action (actions themselves may contain variables) and v is a pattern variable. More precisely a pattern is an abstraction of a trace, where some actions are replaced by variables. They are the basic element used to specify formula in this logic.

The syntax of the logic is defined by the BNF grammar given by Table I.

There is a syntactic restriction on the body of $\nu X.\Phi$ stipulating that any occurrence of X in Φ must occur under the scope of an even number of negations. We also assume that the set of variables in p_2 is included in the set of variables in p_1 (no new variables appearing in p_2). For instance $[x \looparrowright x.y]\nu X.X$ is not a formula since $\{x, y\} \not\subseteq \{x\}$.

From now on, we use the standard abbreviations given by Table II, where $\Phi[\Gamma/X]$ represents the simultaneous replacement of all free occurrences of X in Φ by Γ.

Semantics. Let $\mathcal{L}$ be the set of possible formula of the logic, $\mathcal{V}$ the set of variables, $\mathcal{T}$ the set of traces, Sub the set of substitutions and Env the set of all possible environments in $[\mathcal{V} \to 2^{\mathcal{T}}]$. The semantics of formulas is given by the function $[\![\_]\!]^{\_,\_}_{\_}$:

$$[\![\_]\!]^{\_,\_}_{\_} : \mathcal{L} \times \mathcal{T} \times Sub \times Env \to 2^{\mathcal{T}}$$

This function is inductively defined on the structures of formulas as shown in Table III, where $t_\downarrow$ is the smallest set of traces defined as follows :

$$(i) \quad t \in t_\downarrow$$
$$(ii) \quad t_1.a.t_2 \in t_\downarrow \Rightarrow t_1.t_2 \in t_\downarrow$$

TABLE I.
LOGIC SYNTAX

$\phi ::= X \mid \neg\phi \mid [p_1 \looparrowright p_2]\phi \mid \phi_1 \wedge \phi_2 \mid \nu X.\phi$

TABLE II.
ABBREVIATION OF FORMULA

tt	$\equiv$	$\nu X.X$
ff	$\equiv$	$\mu X.X$
$\langle p_1 \looparrowright p_2\rangle\Phi$	$\equiv$	$\neg[p_1 \looparrowright p_2]\neg\Phi$
$\mu X.\Phi$	$\equiv$	$\neg\nu X.\neg\Phi[\neg X/X]$
$\Phi_1 \vee \Phi_2$	$\equiv$	$\neg(\neg\Phi_1 \wedge \neg\Phi_2)$
$\Phi_1 \to \Phi_2$	$\equiv$	$\neg\Phi_1 \vee \Phi_2$
$\Phi_1 \leftrightarrow \Phi_2$	$\equiv$	$\Phi_1 \to \Phi_2 \wedge \Phi_2 \to \Phi_1$

and $e[X \mapsto U]$ denotes the environment e' defined as follows :

$$e'(Y) = e(Y) \quad \text{if } Y \neq X$$
$$e'(X) = U$$

Environments are used to give a semantics to the formula X and to deal with recursive formula. Substitutions are internal parameters used to give a semantics to the formula $[p_1 \looparrowright p_2]\Phi$. Given an environment e (we start generally with an empty environment) and a substitution σ (we start generally with an empty substitution), we say that a trace t satisfies Φ if $t \in [\![\Phi]\!]^{t,\sigma}_e$.

Intuitively, the trace t satisfies the formula $[p_1 \looparrowright p_2]\Phi$ if for all substitutions σ such that $p_1\sigma = t$, the new trace $p_2\sigma$ (the modified version of the trace t) satisfies the remaining part of the formula (Φ). In this respect, the notation $[p_1 \looparrowright p_2]$ has principally two effects. First, the part p_1 allows us to verify if something has happened somewhere in the trace t. Second, the part p_2 allows us to modify the trace (delete some actions, substitute some actions by others, add some actions) in such a way the remainder of the formula (Φ) will be verified on the modified version of the trace described by p_2.

Example. The following example, given in [1], is helpful to better understand the intuitive meaning of some formula.

Suppose that we want to verify if $t \models \phi$, where t and ϕ are defined as follows :

$$\begin{cases} t = b.a.c.b.d.a \\ \phi = \langle X_1.a.X_2.b.X_3 \looparrowright X_1.X_2.X_3\rangle\langle X_4.b.X_5.d.X_6 \looparrowright X_4.X_5.X_6\rangle tt \end{cases}$$

Then, the verification process involves the following steps :

1) Verify if there exists at least one substitution σ_1 such that the trace t is equal to $(X_1.a.X_2.b.X_3)\sigma_1$. This part is satisfied, since the substitution $\sigma_1 = \{X_1 \mapsto b, X_2 \mapsto c, X_3 \mapsto d.a\}$ fills the required condition.
2) Verify if the second version of the trace defined by $t_1 = (X_1.X_2.X_3)\sigma = b.c.d.a$ satisfies the second part of the formula ($\langle X_4.b.X_5.d.X_6 \looparrowright X_4.X_5.X_6\rangle$)). This part is also satisfied, since there

TABLE III.
DENOTATIONAL SEMANTICS.

$$
\begin{array}{rcl}
[\![X]\!]_e^{t,\sigma} & = & e(X) \\
[\![\neg\Phi]\!]_e^{t,\sigma} & = & t_{\downarrow} - [\![\Phi]\!]_e^{t,\sigma} \\
[\![\Phi_1 \wedge \Phi_2]\!]_e^{t,\sigma} & = & [\![\Phi_1]\!]_e^{t,\sigma} \cap [\![\Phi_2]\!]_e^{t,\sigma} \\
[\![[p_1 \looparrowright p_2]\Phi]\!]_e^{t,\sigma} & = & \{u \in t_{\downarrow} \mid \forall \sigma' :\ p_1\sigma\sigma' = u \Rightarrow p_2\sigma\sigma' \in [\![\Phi]\!]_e^{p_2\sigma\sigma', \sigma'\circ\sigma}\} \\
[\![\nu X.\Phi]\!]_e^{t,\sigma} & = & \nu f \text{ where } \left\{ \begin{array}{rcl} f: \ 2^{\mathcal{T}} & \longrightarrow & 2^{\mathcal{T}} \\ U & \longmapsto & [\![\ \Phi\ ]\!]_{e[X\mapsto U]}^{t,\sigma} \end{array} \right.
\end{array}
$$

exists another substitution $\sigma_2 = \{X_4 \mapsto \epsilon, X_5 \mapsto c, X_6 \mapsto a\}$ such that $t_1 = (X_4.b.X_5.d.X_6)\sigma_2$.

3) Verify if the third version of the trace defined by $t_2 = (X_4.X_5.X_6)\sigma_2 = c.a$ satisfies the final part of the formula ($\sharp t$). This part is also satisfied, since the formula is satisfied by any trace.

We conclude that the trace t satisfies the formula ϕ.

III. TCP/IP ATTACK SPECIFICATION

In [5] we have shown that it is possible to specify a large variety of IP based attacks using our approach. In this paper we are focused on ARP based attacks. We describe hereafter how to detect this kind of attacks by specifying properties using the ADM-logic. To explain the intuitive meaning of formula we give some illustrative examples. Finally we recall some properties characterizing IP based attacks previously defined in [5].

Note that, in the following we consider these two points. All properties are defined in such manner that, the returned result is **true** if the trace contains the attack scenario described by the property, and **false** in the other case.

The terms in bold in formula represent action constants, those in capital letter with subscript annotations (X_1, X_2, etc.) represent pattern variables (do not confuse with the formula variable X), and the terms in small letter represent action parameter variables.

A. ARP based attacks

ARP Cache Poisoning. This attack consists on the corruption of the ARP cache of the victim host (A) by introducing a spurious IP to MAC address mapping. Then if (A) sends out a packet for the host (B) this traffic is diverted to the attacker host (C). To perform this attack, the attacker sends an ARP request to the host (A) having his MAC address and the IP address of the host (B). This request is unicast to avoid suspicion of (B). Then if (A) receives this request it concludes that (B) wants to communicate with it, and so will update its ARP cache entry with the false mapping.

To detect this attack we define the formula ϕ_1 :

$$\neg(\nu X.(\langle X_1.(\mathbf{arp}.eths.ethd.\mathbf{1}.macs.macd).X_2 \looparrowright \epsilon\rangle tt \rightarrow \langle X_1.(\mathbf{arp}.eths.\mathbf{ethd}_{br}.\mathbf{1}.macs.macd).X_2 \looparrowright X_1.X_2\rangle)X)$$

ethd$_{br}$ **= *FF.FF.FF.FF.FF.FF*** (Ethernet broadcast address).

This signature detects an ARP request where the Ethernet destination address is not a broadcast address.

To explain the meaning of this formula, we suppose that we want to verify whether the trace t given in Table IV satisfy $\phi_1 = \neg\phi_1'$.

The first part of the formula $(\langle X_1.(\text{arp}.eths.ethd.1.macs.macd).X_2 \looparrowright \epsilon\rangle tt)$ aims to verify whether the trace contains an ARP request action. If this condition is satisfied, then the second part of this formula $(\langle X_1.(\text{arp}.eths.ethd_{br}.1.macs.macd).X_2 \looparrowright X_1.X_2\rangle X)$ requires that the trace must contain an ARP request action where Ethernet source address is a broadcast address, and if we remove one occurrence of this action, the remaining trace still satisfies the whole formula.

It is clear that the trace t does not satisfy the formula ϕ_1'. Indeed, if we remove the first occurrence of ARP request action from this trace, the remaining trace t' (trace t without the first action) does not satisfy again the formula ϕ_1'. Finally, we conclude that the trace t satisfies the formula ϕ_1.

Note that this kind of attack is possible because there is no coherence control mechanism between the ARP header and the Ethernet header. To guarantee this coherence we specify the following additional properties ϕ_2 and ϕ_3 :

$$\neg(\nu X.(\langle X_1.(\mathbf{arp}.eths.ethd.\mathbf{1}.macs.macd).X_2 \looparrowright \epsilon\rangle tt \rightarrow \langle X_1.(arp.as_1.eths.\mathbf{1}.as_1.macd).X_2 \looparrowright X_1.X_2\rangle)X)$$

$$\neg(\nu X.(\langle X_1.(\mathbf{arp}.eths.ethd.\mathbf{2}.macs.macd).X_2 \looparrowright \epsilon\rangle tt \rightarrow \langle X_1.(\mathbf{arp}.as_1.ad_1.\mathbf{2}.as_1.ad_1).X_2 \looparrowright X_1.X_2\rangle)X)$$

Formula ϕ_2 verify whether the trace contains an ARP request where the Ethernet source address is different

TABLE IV.
TRACE T

Trace	Event
(arp. 11.11.11.11.11.11. FF.FF.FF.FF.FF.FF. 1. 11.11.11.11.11.11. 00.00.00.00.00.00).	ARP request
(arp. 22.22.22.22.22.22. 11.11.11.11.11.11. 2. 22.22.22.22.22.22. 11.11.11.11.11.11).	ARP reply
(arp. 33.33.33.33.33.33. 11.11.11.11.11.11. 2. 33.33.33.33.33.33. 11.11.11.11.11.11).	ARP reply
(icmp. 192.168.1.1. 192.168.1.4. 8.0.0.0).	ICMP echo request
(tcp. 192.168.1.2. 192.168.1.4. 2002. 80. 0.0.0.0.0.1. 0.0).	TCP SYN
(tcp. 192.168.1.4. 192.168.1.2. 80. 2002. 0.1.0.0.0.1. 0.0).	TCP SYN/ACK
(icmp. 192.168.1.4. 192.168.1.1. 0.0. 0.0).	ICMP echo reply
(arp. 33.33.33.33.33.33. 22.22.22.22.22.22. 1. 33.33.33.33.33.33. 00.00.00.00.00.00).	ARP request (Unicast)
(arp. 22.22.22.22.22.22. 33.33.33.33.33.33. 2. 22.22.22.22.22.22. 33.33.33.33.33.33).	ARP reply

from the source address in the ARP message, and formula ϕ_3 verify if there is an ARP response where the Ethernet source/destination address is different from the source/destination address in the ARP message.

ARP Spoofing. The ARP works as follows. A host (A) that wants to resolve an IP address of a host (B) broadcast an ARP request. When receiving this packet host (B) replies by sending an ARP response containing its MAC address. Host (A) can subsequently communicate directly with host (B) using the MAC address. However, host (C) (the attacker) can also reply to the ARP request that will have as a consequence the update of the ARP table of the host (A) with the false mapping extracted from the ARP response sent by host (C).

To detect this attack we propose a formula ϕ_4 that detects an ARP response that are not preceded by an ARP request (i.e. for each ARP response we need to have its corresponding ARP request).

$$\neg(\nu X.(\langle X_1.(\textbf{arp}.eths.ethd.\textbf{2}.macs.macd).X_2.\ \neg\!\!\rightarrow \epsilon\rangle tt \rightarrow \langle X_1.(\textbf{arp}.as_1.ethd.\textbf{1}.as_1.macd).X_2.(arp.eths.as_1.\textbf{2}.macs.as_1).X_3 \neg\!\!\rightarrow X_1.X_2.X_3\rangle X))$$

Note that trace t (see Table IV) satisfy the formula ϕ_4.

B. IP based attacks

Crafted Packets. Abnormal packets [6] are used by attackers to probe networks or to crash systems. For example, there are several flag combinations that can be classified as abnormal like SYN/FIN packet and Null packet. To detect such attacks, we define the following formulas :

$$\langle X_1.(\textbf{tcp}.as.ad.ps.pd.flags_1.frags).X_2 \neg\!\!\rightarrow \epsilon\rangle tt$$

$flags_1$ = $f_1.f_2.f_3.f_4.$***1.1*** (flags SYN and FIN are set).

$$\langle X_1.(\textbf{tcp}.as.ad.ps.pd.\textbf{flags}_1.frags).X_2 \neg\!\!\rightarrow \epsilon\rangle tt$$

flags$_1$***= 0.0.0.0.0.0*** (Null packet)

Land attack. This attack [7] consists in sending a spoofed packet with the SYN flag set, the same source and destination addresses and also the same source and destination ports.

To detect this attack we define the following formula :

$$\langle X_1.(\textbf{tcp}.a.a.p.p.flags_1.frags).X_2 \neg\!\!\rightarrow \epsilon\rangle tt$$

$flags_1$ = $f_1.f_2.f_3.f_4.f_5.$***1*** (flag SYN is set).

Smurf attack [8]. Attackers forge an ICMP echo request message (ping) that contains the spoofed source address of the victim, and then direct this packet to IP broadcast addresses. The result is that the victim is flooded by several ICMP echo reply (pong). To detect this attack we define the following formula :

$$\neg(\nu X.(\langle X_1.(\textbf{icmp}.as.ad.\textbf{0.0}.frags).X_2.\ \neg\!\!\rightarrow \epsilon\rangle tt \rightarrow \langle X_1.(\textbf{icmp}.as.ad.\textbf{8.0}.frags_1).X_2.(\textbf{icmp}.ad.as.\textbf{0.0}.frags_2).X_3 \neg\!\!\rightarrow X_1.X_2.X_3\rangle X))$$

$ping =$***icmp***$.as.ad.$***8.0***$.frags_1$
$pong =$***icmp***$.as.ad.$***0.0***$.frags_2$

This signature considers as an attack any trace containing an ICMP echo reply (pong) that is not preceded by an ICMP echo request (ping). We also define the following formula to detect an ICMP echo request (ping) having as destination address a broadcast address.

$$((\langle X_1.(\textbf{icmp}.as.ab_1.\textbf{8.0}.frags).X_2 \neg\!\!\rightarrow \epsilon\rangle tt)\vee (\langle X_1.(\textbf{icmp}.as.ab_2.\textbf{8.0}.frags).X_2 \neg\!\!\rightarrow epsilon\rangle tt))$$

ab_1 = $b_1.b_2.b_2.$***0*** (broadcast address)
ab_2 = $b_1.b_2.b_2.$***255*** (broadcast address)

SYN Flooding attack [9]. The attack occurs when many SYN packets are sent to the target in order to fill its backlog queue with pending connections. From this time, the target machine ignores all further connections attempts (DoS attack). To prevent this attack we define the following formula :

$$\nu X.(\langle X_1.(\textbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_1.frags).X_2.(\textbf{tcp}.as_2.ad_2.ps_2.pd_1 flags_1.frags).X_3.(\textbf{tcp}.as_3.ad_3.ps_3.pd_1.flags_1.frags).X_4 \neg\!\!\rightarrow \epsilon\rangle tt \rightarrow (\langle X_1.(\textbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_1.frags).X_2.(\textbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_2.frags).X_3 \neg\!\!\rightarrow X_1.X_2.X_3\rangle X \vee \langle X_1.(\textbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_1.frags).X_2.(\textbf{tcp}.ad_1.as_1.pd_1.ps_1.flags_3.frags).X_3 \neg\!\!\rightarrow X_1.X_2.X_3\rangle X))$$

$flags_1$ = $f_1.f_2.f_3.f_4.f_5.\mathbf{1}$ (SYN flag is set)
$flags_2$ = $f_1.\mathbf{1}.f_2.f_3.f_4.f_5$ (ACK flag is set)
$flags_3$ = $f_1.f_2.f_3.\mathbf{1}.f_4.f_5$ (RST flag is set)

This signature states that if the number of half-open connections is over a fixed threshold (threshold = 3) then considers this as an attack. Half-open connection is detected if there are SYN packets not followed by ACK or RST packets.

Echo-Chargen Loop attack. This attack [10] is based on UDP. It consists in setting up a loop between the echo and chargen ports. To detect this attack, we propose a formula that detects packets having a source or a destination port set to 7 and 19.

$$\langle X_1.(\mathbf{udp}.as.ad.\mathbf{7}.\mathbf{19}).X_2 \looparrowright \epsilon\rangle tt \vee \langle X_1.(\mathbf{udp}.as.ad.\mathbf{19}.\mathbf{7}).X_2 \looparrowright \epsilon\rangle$$

Scan. There are several techniques for surveying ports on which a target machine is listening. Among these techniques, we find TCP FIN scan [11]. It is a furtive scan that detects closed ports to deduce opened ones. This scan is carried out by sending FIN packets without opening connections. If the port is closed, the target machine sends RST packet. Otherwise, no answer will be turned back. For this kind of attack, we define the following formula, which detects FIN packets that are not preceded by SYN packets. Furthermore, a one-to-one association between SYN and FIN should be established (i.e. for each SYN we need to have its corresponding FIN).

$$\neg(\nu X.(\langle X_1.(\mathbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_2.frags).X_2 \looparrowright \epsilon\rangle \rightarrow \langle X_1.(\mathbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_1.frags).X_2.(\mathbf{tcp}.as_1.ad_1.ps_1.pd_1.flags_2.frags).X_3 \looparrowright X_1.X_2.X_3\rangle)X)$$

$flags_1$ = $f_1.f_2.f_3.f_4.f_5.\mathbf{1}$ (SYN flag is set)
$flags_2$ = $f_1.f_2.f_3.f_4.\mathbf{1}.f_5$ (FIN flag is set)

Notice that this kind of properties could not specified using LTL for example.

Ping of Death [12]. The attacker sends a stream of ICMP echo request fragments that, if assembled, they create an IP datagram longer than 65535 bytes. To prevent this attack we suggest the following formula that detects fragmented pings :

$$\neg(\nu X.(\langle X_1.(\mathbf{icmp}.as.ad.\mathbf{8}.\mathbf{0}.frags).X_2 \looparrowright \epsilon\rangle tt \rightarrow \langle X_1.(\mathbf{icmp}.as.ad.\mathbf{8}.\mathbf{0}.\mathbf{frags}_1).X_2 \looparrowright X_1.X_2\rangle)X)$$

$\boldsymbol{frags}_1$ = ***0.0*** (non fragmented packet)

Tiny fragments and fragment overlapping attacks [12]. These attacks consist in requesting a TCP connection fragmented into two IP packets. To detect them, we proceed in the same manner as for the Ping of Death attack, i.e., we look for fragmented SYN packets.

$$\neg(\nu X.(\langle X_1.(\mathbf{tcp}.as.ad.ps.pd.flags_1.frags).X_2 \looparrowright \epsilon\rangle tt \rightarrow \langle X_1.(\mathbf{tcp}.as.ad.ps.pd.flags_1.\mathbf{frags}_1).X_2 \looparrowright X_1.X_2\rangle)X)$$

$flags_1 = f_1.f_2.f_3.f_4.f_5.\mathbf{1}$ (SYN is set)
$\boldsymbol{frags}_1$ = ***0.0*** (non fragmented packet)

IV. Implementation

Another important advantage of using the ADM-logic comes from it tableau-based proof system that has been proved, in [1], to be sound and complete with respect to the denotational semantics. In fact, this tableau-based semantics leads to a local model checking which is useful for an efficient implementation.

In this section, we give an overview of our IDS prototype, written in Java and called GTM-system (after its authors Ghorbel, Talbi and Mejri) which is based on the ADM-logic. The GTM-system can be seen as a model checker that verifies user properties against models. In this paper, we focused on models that abstract audit trail, but the system can be used to verify any sequence of events (linear models).

A. Tableau-Based Proof System

This verification is based on the tableau-based proof system given in the Table V. The idea behind the tableau rules is to capture in a deductive way whether a trace t satisfies a formula ϕ or not. The proof rules operate on sequents of the form $H, b, e, \sigma \vdash t \in \phi$, where H is a mapping in $[\mathcal{V} \rightarrow 2^{\mathcal{T}}]$, b is a variable in $\{\epsilon, \neg\}$, σ a substitution, e an environment, t a trace and ϕ a formula.

A sequent θ has a successful tableau if there exists a finite tableau having θ as a root and all its leaves are successful. A leaf θ is successful when it meets one of the following conditions :

- θ = (H, ϵ, e, $\sigma \vdash t \in X$) and t $\in$ e(X).
- θ = (H, $\neg$, e, $\sigma \vdash t \in X$ and $t \in e(X)$.
- θ = (H, ϵ, e, $\sigma \vdash t \in \nu X.\phi$) and $t \in H(X)$.
- θ = (H, ϵ, e, $\sigma \vdash t \in [p_1 \looparrowright p_2]\phi$) and $\{\sigma' | p_1\sigma\sigma' = t\} = \emptyset$.

It follows that an unsuccessful leaf θ, will denote sequents that are not successful leaves and there are no rules of the tableau system that could be applied on them. More precisely, a sequent θ is an unsuccessful leaf, if one of the following conditions holds :

- θ = (H, ϵ, e, $\sigma \vdash t \in X$) and t $\in$ e(X).
- θ = (H, $\neg$, e, $\sigma \vdash t \in X$ and $t \in e(X)$.
- θ = (H, $\neg$, e, $\sigma \vdash t \in \nu X.\phi$) and $t \in H(X)$.
- θ = (H, $\neg$, e, $\sigma \vdash t \in [p_1 \looparrowright p_2]\phi$) and $\{\sigma' | p_1\sigma\sigma' = t\} = \emptyset$.

For further details about the rules of the Table V, see [1].

B. GTM-System Architecture

An overview of the GTM-System architecture is presented in Figure 1. It is composed of three modules : compiler, abstracter and checker.

Compiler. It checks the syntax of the given formula and reports errors, if they exist, to the end-user. This procedure is implemented by means of Flex and Bison tools [13]. If the properties are free of syntax errors, they will be

TABLE V.
TABLEAU-BASED PROOF SYSTEM

$$R_{\neg} \quad \frac{H, b, e, \sigma \vdash t \in \neg\phi}{H, \neg b, e, \sigma \vdash t \in \phi}$$

$$R_{\wedge} \quad \frac{H, b, e, \sigma \vdash t \in (\phi_1 \wedge \phi_2)}{H, b_1, e, \sigma \vdash t \in \phi_1 \qquad H, b_2, e, \sigma \vdash t \in \phi_2} \qquad b_1 \times b_2 = b$$

$$R_{\nu} \quad \frac{H, b, e, \sigma \vdash t \in \nu X.\phi}{H[X \mapsto H(X) \cup t], b, e, \sigma \vdash t \in \phi[\nu X.\phi/X]} \qquad \text{t} \in \text{H(X)}$$

$$R_{[]} \quad \frac{H, b, e, \sigma \vdash t \in [p_1 \looparrowright p_2]\phi}{H, b_1, e, \sigma_1 \circ \sigma \vdash p_2\sigma\sigma_1 \in \phi \quad \ldots \quad H, b_n, e, \sigma_n \circ \sigma \vdash p_2\sigma\sigma_n \in \phi} \qquad C_2$$

$$C_2 = \left(\begin{array}{c} \{\sigma_1, ..., \sigma_n\} = \{\sigma' | p_1\sigma\sigma' = t\} \neq \emptyset \\ and \\ b_1 \times \ldots\ldots\ldots\ldots \times b_n = b, n > 0 \end{array} \right)$$

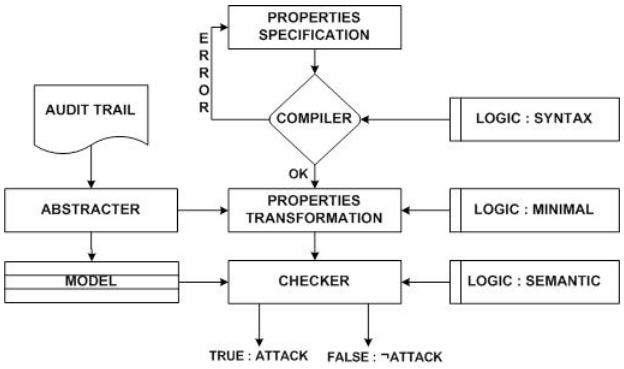

Figure 1. GTM-System Interface

transformed according to abbreviations given by Table II. The idea behind these abbreviations, is to implement only a few numbers of operators (those of the ADM-logic), the rest can be derived from macros.

Abstracter. It allows to abstract the audit file by keeping only relevant events specified by a given filter. By default, the filter extract from an audit file (tcpdump file [14]) only ARP, ICMP, TCP and UDP packets and then structure them to form the final trace (model) following the action format defined in section 2. The extraction process is implemented using the jpcap package [15].

The role of the abstracter module is very important. The fact that it is parameterizable (through the use of filter), we can easily extend our model to specify new signatures, such those related to application layer protocols (HTTP, SMTP, DNS, etc.). For instance, suppose we want specify properties related to HTTP protocol. This can be done by simply adding this protocol in the abstracter filter and specifying the HTTP action format. Thus, if the abstracter find a web packet (TCP packet having destination port equal to 80, 8080), then it will generate an HTTP event from the packet data field.

Checker. This is the main part of the system and it implements the tableau-based proof system of the ADM-logic. In other word, this part implements the satisfaction relation "$\models$" of the model checker. It takes a model $\mathcal{M}$ (trace) and a formula ϕ, and verify whether $\mathcal{M} \models \phi$ or not. At the end of this stage, the system give us the detection results : **true** indicates that the trace contains the attack-evidence described by the specified formula.

To show how we can verify if a formula is satisfied or not by a trace using the tableau-based proof system, let us give a concrete example :

Let $t = a.b.c$ be a trace where the actions a, b and c represent respectively, SYN packet, SYN FIN packet and SYN ACK packet.

TABLE VI.
PROOF TREE

$$R_{\neg} \quad \frac{\emptyset, \epsilon, \emptyset, \emptyset \vdash a.b.c \in \neg[X_1.b.X_2 \looparrowright X_1.X_2]\neg\nu X.X)}{\emptyset, \neg, \emptyset, \emptyset \vdash a.b.c \in [X_1.b.X_2 \looparrowright X_1.X_2]\neg\nu X.X}$$

$$R_{[]} \quad \frac{}{\emptyset, \neg, \emptyset, \sigma \vdash a.c \in \neg\nu X.X}$$

$$R_{\neg} \quad \frac{}{\emptyset, \emptyset, \emptyset, \sigma \vdash a.c \in \nu X.X}$$

$$R_{\nu} \quad \frac{}{[X \mapsto \{a.c\}], \emptyset, \emptyset, \sigma \vdash a.c \in \nu X.X}$$

Consider now the formula $\phi = \langle X_1.b.X_2 \looparrowright X_1.X_2\rangle tt$ which allows to detect if the trace t contains a SYN FIN packet (abnormal packet [6] used by nmap [16] to probe networks).

Proving that the formula ϕ is satisfied by the trace t, amounts to show that the sequent $\emptyset, \epsilon, \emptyset, \emptyset \vdash a.b.c \in \langle X_1.b.X_2 \looparrowright X_1.X_2\rangle tt$ leads to a successful leaf.

We know that :

$\langle X_1.b.X_2 \looparrowright X_1.X_2\rangle tt \equiv \neg[X_1.b.X_2 \looparrowright X_1.X_2]\neg\nu X.X$

Now, let σ denote the set $\{X_1 \mapsto a, X_2 \mapsto c\}$.

The proof associated with the sequent $\emptyset, \epsilon, \emptyset, \emptyset \vdash a.b.c \in \neg[X_1.b.X_2 \looparrowright X_1.X_2]\neg\nu X.X$ is given in Table VI.

The leaf of the derivation sequence is a successful sequent because it satisfies the condition :

$\theta = (H, \epsilon, e, \sigma \vdash t \in \nu X.\phi)$ and $t \in H(X)$

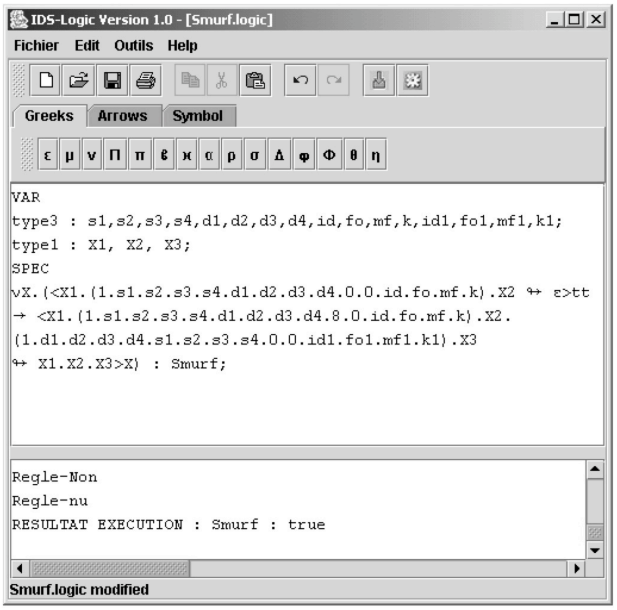

Figure 2. GTM-System Interface

We conclude that the trace t satisfies the formula ϕ.

C. GTM-System Interface

As shown by Figure 2, the GTM-system has a simple and user-friendly interface allowing the end-user to specify his security policy and an audit file and then the tool make the analysis. First, the user defines his variables[1] with the clause VAR. Then, properties using these variables can be specified using the reserved word SPEC. A name can be attributed to a property so that it could be referred later in other formula or in the output of the analysis.

V. RELATED WORK

Many works presented in the literature are dedicated to the intrusion detection issue. In this section we are limited to works based on formal approaches since they are closely related to our work. Such approach consists on the use of a given logic for the specification of properties characterizing attack scenario. These properties are then checked against a model (linear or arborescent model) that abstract the audit trail (network, host or application based audit data).

The choice of the appropriate logic depends on the security objectives. As example, the logic used in [17] is especially conceived for the specification of properties characterizing polymorphic shellcodes (ADMmutate [18], Clet [19], etc.). This logic combines two temporal logics LTL (Linear-Time Temporal Logic) and CTL (Computational Tree Logic) [4]. In [2] author propose a new logic to detect variants of known attacks. This logic is inspired from LTL to which some new features are added.

Although LTL and CTL logics are simple and easily comprehensible by users, it is less expressive than ADM logic which is endowed with several interesting features for intrusion detection such as modalities, linearity and recursive formulas. For example, some formulas defined in section 3 such as ϕ_1 and ϕ_2 could not be specified using LTL.

[1]Notice that variables are sorted : type1= pattern variables, type2= event variables and type3= event parameter variables.

VI. CONCLUSIONS

In this paper, we have introduced a model-checker called GTM-system that can be used as an intrusion detection system. In fact, the GTM-system is based on ADM-Logic which is appropriate to specify a large variety of attacks and in particular those based on Internet protocols (TCP, UDP, ICMP and ARP). Thanks to the tableau-based proof system of ADM, the implementation of the GTM-system was made simple and efficient.

In addition, this work can be improved by studying these two considerations. Firstly, we can specify properties related to application layer based attacks (attacks based on protocols such HTTP, DNS, SMTP, etc.). This can be done by specifying the action format of these protocols in our model, and by making some modifications to the abstracter. Secondly, we can evaluate our properties in terms of false positives and false negatives. This evaluation can be done either by using the arsenal of tools available in Internet ([20], [16]), or by having recourse to an attack description language such as STATL [21].

REFERENCES

[1] K. Adi, M. Debbabi, and M. Mejri, "A New Logic For Electronic Commerce Protocols," *Theor. Comput. Sci.*, vol. 291, no. 3, pp. 223–283, 2003.

[2] P. L. Lesperance and M. Mejri, "Detecting Variants of Known Attacks Using Temporal Logic," in *Proceedings of a Workshop on Practice and Theory of Access Control Technologies (WPTACT 2005)*, 2005.

[3] C. Stirling, "Modal and Temporal Logics for Processes," in *Banff Higher Order Workshop*, 1995, pp. 149–237.

[4] M. Huth and M. Ryan, *Logic in Computer Science : Modelling and Reasoning about Systems*. Cambridge, England : Cambridge University Press, 2000.

[5] M. B. Ghorbel, M. Talbi, and M. Mejri, "Specification and Detection of TCP/IP Based Attacks Using the ADM-Logic," in *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*. IEEE Computer Press, 2007.

[6] K. K. Frederick, "Abnormal IP Packets," http ://www.securityfocus.com/infocus/1200, 2000.

[7] m3lt, "The LAND attack (IP DOS)," http ://insecure.org/sploits/land.ip.DOS.html, 1997.

[8] "CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks," http ://www.cert.org/advisories/CA-1998-01.html.

[9] Daemon9, "Project Neptune," Phrack Magazine, http ://www.phrack.org/archives/48/P48-13.

[10] "CERT Advisory CA-1996-01 UDP Port Denial-of-Service Attack," http ://www.cert.org/advisories/ CA-1996-01.html.

[11] U. Maimon, "Port Scanning without the SYN flag," Phrack Magazine, http ://www.phrack.org/archives/49/P49-15.

[12] E. Detoisien, "Les Attaques Externes," Misc Magazine.

[13] "The Lex & YACC Page," http ://dinosaur.compilertools.net.

[14] "Tcpdump," http ://www.tcpdump.org/.

[15] "Network Packet Capture Library," http ://jpcap.sourceforge.net/.

[16] Fyodor, "The Art of Port Scanning," Phrack Magazine, 1997.

[17] M. Talbi, M. Mejri, and A. Bouhoula, "New Temporal Logic for Polymorphic Shellcode Detection," Higher School of Communication of Tunis (SUP'COM), Tech. Rep., 2007.

[18] K2, "ADMmutate," http ://www.ktwo.ca/ADMmutate-0.8.4.tar.gz.

[19] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. S. V. Underduk, "Polymorphic Shellcode Engine Using Spectrum Analysis," Phrack Magazine, 2003.

[20] F. Raynal, "Arp-sk," http ://sid.rstack.org/arp-sk/.

[21] S. Eckmann, G. Vigna, and R. Kemmerer, "STATL : An Attack Language for State-based Intrusion Detection," *Journal of Computer Security2*, vol. 10, no. 1/2, pp. 71–104, 2002.

Mehdi Talbi was born in Tunisia. He is currently a Ph.D. student in cotutelle between the Higher School of Communication (SUP'COM) in Tunisia and the High School of Electrical Engineering (SUPELEC) in France. He received his Master degree in telecommunication and computer science from the Higher School of Communication (SUP'COM) in 2006 and his Engineering degree in network and computer science from the National Institute of Applied Sciences and Technologies (INSAT), Tunisia, in 2003.

He is currently an Assistant Professor of Computer Science at INSAT, Tunisia. His research interests include intrusion detection systems and cryptographic protocol verification.

Meriam Ben Ghorbel-Talbi was born in Tunisia. She is currently a Ph.D. student in cotutelle between the Higher School of Communication (SUP'COM) in Tunisia and the National School of Telecommunication (ENST-Bretagne) in France. She received her Master degree in computer science from the National School of Computer Science (ENSI), Tunisia, in 2005 and her Engineering degree in network and computer science from the National Institute of Applied Sciences and Technologies (INSAT), Tunisia, in 2003.

She is currently an Assistant Professor of Computer Science at INSAT, Tunisia. Her research interests include intrusion detection systems and access control policies.

Mohamed Mejri Mohamed Mejri is a professor in Computer Science Department at Laval University, Quebec, Canada. He was born in Tunisia where he has received his Engineering degree in computer science from the National School of Computer Science (ENSI). His master degree and Ph. D. degree are in computer science from Laval University, Quebec, Canada. His research interest includes computer security and formal methods.