A Prototype for Liunx/Unix Intrusion Detection:Approach by Behavior Specification

Ines BEN TEKAYA, Béchir AYEB
Unité de recherche PRINCE
Faculté des Sciences
5000 Monastir, Tunisia
bentekaya.ines@voila.fr, ayeb_b@yahoo.com

ABSTRACT: *This paper describes literature works in intrusion detection field. After that, we propose an intrusion detection method in Linux/Unix commands using supervisor synthesis. This method was applied to distinct normal user behavior from intruders behavior. The main features of this work are twofold. It exploits supervisor synthesis in the intrusion detection field. It presents our approach by behavior specification. Two advantages characterize our proposed algorithm for detection. The first advantage is that the algorithm result is a structure. The second advantage is the way of searching faults or intrusions.*

Keywords: Intrusion Detection, Observed User's Behavior, Specification Model, Linux/Unix commands, Supervisor Synthesis

Received: 18 May 2013, Revised 28 June 2013, Accepted 5 July 2013

1. Introduction

The intrusion field was introduced by Anderson. It was defined as an attempt or a threat to be the potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable [1]. The difference between intrusion and attack consists of the fact that intrusion is a malicious, externally or internally induced fault resulting from an attack that has succeeded in exploiting vulnerability, while a fault is the adjudged or hypothesized cause of an error, the cause of which is intended to be avoided or tolerated. An attack is a malicious technical interaction fault aiming to exploit vulnerability as a step towards achieving the final aim of the attacker [2].

A statistical study shows that 98% of enterprises have a firewall to be protected from external attacks; however, 80% of attacks came from internal users [3]. Detecting internal normal user behavior is a difficult problem because a user can have much dynamic behavior and it will be almost impossible to create user profiles that determine the normal behavior. Using a system to distinct normal user from intruders is necessary. This system is called Intrusion Detection System (IDS). It is defined as a security technology attempting to identify and isolate computer systems intrusions [4].

We choose to work with Unix/Linux operating system because in people's minds, if it is non-Windows, it is secure [5]. This hypothesis will be countered here. More details for Unix/Linux system can be found in [6].

The literature on detection using Linux/Unix commands offers a variety of methods. Despite their diversity, their common objective is to distinguish between a normal behavior and an intrusive behavior. The present work falls mainly within the model approach. The data are not based in the past event but they compose a model. It is a theoretical representation of a system which is composed of elements and relation.

The reminder of the paper is organized as follows. Section 2 deals with intrusion background. In section 3, we describe our behavior model construction. In section 4, we present our method which exploit supervisor synthesis in the intrusion detection field. In section 5 we will draw our conclusions and plan for future work.

2. Intrusion Background

The next subsections summarize attacks topology, some dataset used in the literature for intrusion detection and show detection methods using Unix commands.

2.1 Attacks Topology

Attacks take several forms to break one or more of the security properties. They can be grouped according to their functionality as described in the following subsections [7]:

2.1.1 Gathering Security-relevant Information

Before experiencing an attack, a hacker tries to obtain necessary information that is probably sensible about the targeted system, which can be employed later to obtain access to this system. Useful information can be obtained by different ways such as network scanning and vulnerability scanning or even by using public search engines such as Google or social engineering methods.

2.1.2 Access Gain Attacks

With informati on gathered by the above methods, attackers try to obtain a privileged access on a system by exploiting vulnerabilities in the services or the applications installed on this system or a bad configuration of the network. This kind of attacks primarily grants unauthorized access to the targeted system. For example, one of the configuration problems is the use of weak passwords in systems where a bad policy of password definition allows users to choose simple and easy guessable passwords. Otherwise, an attacker can use cracking tools such as "*john the ripper*" [8] to obtain passwords by bruteforce. Buffer-overflow attacks are another example that allows attackers to execute arbitrary code on the targeted hosts.

2.1.3 Denial of service (DoS)

DOS attacks are designed to overload or disable the capabilities of a machine or a network, and thereby render it unusable or inaccessible. An example of denial of service is a fork bomb. It works by creating a large number of processes very quickly in order to saturate the available space in the list of processes kept by the computer's operating system. If the process table becomes saturated, no new programs may start until another process terminates.

2.1.4 Malware Attacks

This category of attacks can result in damages as simple as displaying a simple flicker to catastrophic damages such as completely formatting hard disks. It groups virus, worm, Trojan horse, spyware, rootkit [9] and spam.

2.2 Detection Using UNIX Commands

The object of intrusion can be files, data bases, network connection, Input/output systems or commands Linux/Unix. In this paper we are interested about intrusion using Linux/Unix commands because it can characterize user behaviour more efficiently than other object. The followings paragraphs present some works about methods using Unix commands. These works are classified into two classes: the class of intrusion detection and the class of masquerade detection.

Ilgun, et al. present the state transition analysis method [10] [11]. They used the known Unix intrusion to create a penetration scenario. A penetration is viewed as a sequence of actions performed by an attacker that leads from some start state on a system to a target compromised state, where a state is a snapshot of the system representing the values of all volatile, semi-permanent and permanent memory locations on the system. The start state corresponds to the state of the system just prior to the execution of the penetration. The compromised state corresponds to the state resulting from the completion of the penetration. Between the start and compromised states are one or more intermediate state transitions that an attacker performs to achieve the compromise.

Another method is based on sequence matching. The incoming stream event is segmented into overlapping fixed length sequences. The choice of the sequence length, l, depends on the profiled user. In practical, it's fixed to the value $l = 10$ in the SEA dataset [12]. Each sequence is then treated as an instance in an l-dimensional space and is compared to the known profile. The profile is a set, $\{T\}$, of previously stored instances and comparison is performed between all $y \in \{T\}$ and the test sequence via

a similarity measure. Similarity is defined by a measure, *Sim* (x, y), which makes a point-by-point comparison of two sequences, x and y, counting matches and assigning greater weight to adjacent matches.

The maximum of all similarity values computed forms the score for the test command sequence. Since these scores are very noisy, the most recent 100 scores are averaged. If the average score is below a threshold an alarm is raised. The threshold is determined based on the quantiles of the empirical distribution of average scores [13].

Another method, used statistical method, is called uniqueness. It is based on the idea that commands not previously seen in the training data may indicate an attempted masquerade. Uniquely used commands account for 3% of the data. A command has popularity 'i' if exactly I users use that command. They group the commands such that each group contains only commands with the same popularity. They define a test statistic that builds on the notion of unpopular and uniquely used commands. They assign the same threshold to all users. This threshold is estimated via cross validation: They split the original training data in the SEA dataset into two data sets of 4000 and 1000 commands. Using the larger data set as training data, they assign scores for the smaller one. This is repeated five times, each time assigning scores to a distinct set of 1000 commands. They set the threshold to the 99th percentile of the combined scores across all users and all five cross validations. For their data, the resulting threshold is 0.2319 [12] [14].

Another method is called Bayes 1-Step Markov Model. It is proposed by Schonlau, et al. The authors use the information of 1-step command transition probabilities. They build transition matrices for each user's training and testing data. The detector triggers the alarm when there is a considerable difference between the training data transition matrix and the testing data matrix. This technique was the best performer in terms of correct detections, but failed to get close to the desired false alarm rate [12].

Maxion use Naive Bayes classifiers and detect masqueraders by looking at the classifiers misclassification behavior [15]. This method use command occurrence probability distribution modeling the UNIX sequence. The goal of the training procedure is to establish profiles of self and nonself, and to determine a decision threshold for discriminating between examples of self and nonself. For each User X in the SEA dataset, a model of Not X can also be built using training data from all other victims. The probability of the test sequence having been generated by Not X can then be assessed in the same way as the probability of its having been generated by User X. The larger the ratio of the probability of originating with X to the probability of originating with Not X, the greater the evidence in favor of assigning the test sequence to X. The exact cut-off for classification as X, that is the ratio of probabilities below which the likelihood that the sequence was generated by X is deemed too low, can be determined by a cross-validation experiment during which probability ratios for sequences which are known to have been generated by self are calculated, and the range of values these legitimate sequences cover is examined.

2.3 Limitations in Existing Methods

The intrusion detection method in Linux/Unix commands using a model seeks to improve on some of limitations that the authors observed in the existing methods. This section briefly identifies some of their characteristics.

The major weakness of these methods is that they depend on aggregative, training or experimental past data. The results of statistical methods are closed to the training data while the result of state transition analysis method is depend with the defined penetrations attacks which are non valuable now.

Lastly, they cannot make difference between the orders of commands in the sequence used. The statistical methods are based on the command frequency while a state transition analysis method can't detect the attacks based in frequency such as deny of service.

In the following, we focus on these limitations to present our approach by Behavior Specification.

3. Behavior Model Condtruction

3.1 Preliminaries

We start with some general definitions.

An automaton G is represented formally by a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where, Q is the state set, Σ is a finite alphabet of symbols, δ is the transition function, q_0 is the start state and Q_m are the marked states.

Consider an observed behavior modeled by the finite-state automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q and Σ are the finite state and event sets, δ is the transition function, q_0 the start state and Q_m the marked states.

The specifications model or the user behavior model uses a single automaton. Two automatons $A1$ and $A2$ defined respectively by: $A1 = (Q_1, \Sigma, \delta_1, q_{0,1}, X_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, X_2)$.

The automaton A resulting from the union of automatons $A1$ and $A2$ is defines by $A = (Q, \Sigma, \delta, q, X)$ where, $Q = Q_1 \times Q_2$, $\delta((q1, q2), a) = \{ (q_1', q_2') \mid , q_1' \in \delta_1 (q_1, a)$ and $q_2' \in \delta_2 (q_2, a)$, $q = q_{0,1} \times q_{0,2}$, $X = Q_1 \times X_2 \cup X_1 \times Q_2$

3.2 Modeling user behavior

Let us first formulate model user behavior on three steps. First, we should define for every command its rank, if it is iterative and the interconnected commands:

• **A command of rank 0:** is the command which no depends on other command

• **A command of rank 1:** is the command which comes from a command of rank 0.

Generalization: a command of rank n ($n > 0$) is a command which comes from a command of rank $n - 1$.

• An iterative command if it can run several times successively

• An interconnected command: when a command 1 has a relation between a command 2. For example when we use FTP command, we use after that get or put commands.

Second, for a command of rank 0, we create a new transition, labeled the command number, from state 0 (the start state).

Third, we test if a command is an iterative command, we loop in the same state. Else, we create a new state only if this state doesn't exist. So we use a search function to verify the state existence. We create a transition only if there is a relation between commands (inter-connected commands).

As an illustrative example, let be the specification model 1: command 1 command 2 command 3. Let be the specification model 2: command 1 command 4 command 5.

We have: rank (command 1) = 0, rank (command 2) = 1, rank (command 3) = 2, rank (command 4) = 1 and rank (command 5) = 2. Also: interconnected (command 1) = {command 2, command 4}, interconnected (command 2) = {command 3} and interconnected (command 4) = {command 5},

By adopting the union formula of two automatons, the result specification model is illustrated by figure 1. It is formed by six states: The state 0 corresponds to state 0 of the specification model 1 and state 0 of specification model 2.

The state 1 corresponds to state 1 of the specification model 1 and state 1 of specification model 2.

The state 2 corresponds to state 0 of the specification model 1 and state 1 of specification model 2.

The state 3 corresponds to state 3 of the specification model 1 and state 1 of specification model 2.

The state 4 corresponds to state 1 of the specification model 1 and state 2 of specification model 2.

The state 5 corresponds to state 1 of the specification model 1 and state 3 of specification model 2.

3.3 Modeling The Observed Behavior

A quick analysis shows that we have to distinguish two cases. That is, whether commands are order rank-sensitive or not.

The first case appears rarely when commands must be executed in that order. The observed's user behavior is model by an automaton. The transitions will be the commands and states are incremental.

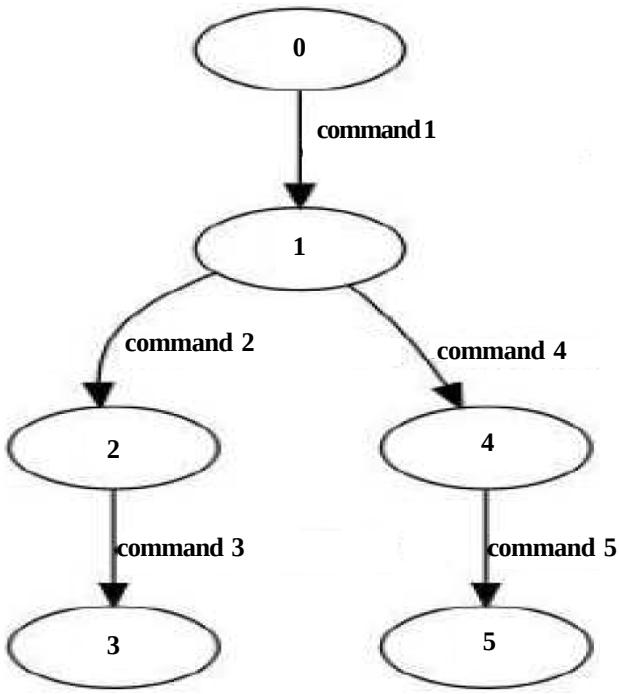

Figure 1. An example of specification result model

The second case consists in ordering the commands typed by the user according to their ranks and their interrelation. We respect the order in which the user set these commands. So, we have a commands processing phase to select their ranks and the interconnected commands.

After that, we test for every command if it is of rank 0. If that is, we create a new state and a transition labeled by the digitized command. Else, two cases appear:

• **The current command is not the first command typed by the user:** we look for an inter-connected command. In case of existence of this command, we add a transition from the state of this command to a new state which we create. In case of non-existence of an interconnected command, we create one or several transitions labeled not defined (nd) until we have the rank of this command.

• **The current command is the first command typed by the user, but it is not of rank 0:** we create one or several transitions labeled not defined (*nd*) until we have the rank of this command.

4. Observed Behavior Analysis and Anomaly Detection

4.1 Preliminaries

The synchronous composition $Syn = G \,||\, K$ is defined by $(Q \times X, \Sigma_1 \cup \Sigma_2, \delta \times \xi, q_0 \times x_0, Q_m \times X_m)$ Where, $Q \times X$: states set, $\Sigma_1 \cup \Sigma_2$: Alphabet of *Syn*, $q_0 \times x_0$: Start state, $Q_m \times X_m$: Marked states set, $\delta \times \xi$: The transition function defined by:

$$(\delta \times \xi)((q, x), \sigma) = (q', x') \; if$$

$$\delta(q, \sigma) = q'! \; and \; \xi(x, \sigma) = x'! \qquad (1)$$

$$(\delta \times \xi)((q, x), \sigma) = (q', x')! \; with \; \delta \in \Sigma_1 \setminus \Sigma_2 \qquad (2)$$

$$(\delta \times \xi)((q, x), \sigma) = (q, x')! \; with \; \xi \in \Sigma_2 \setminus \Sigma_1 \qquad (3)$$

Equation 1 defines the transition function $\delta \times \xi$ when a new state, for q and for x, was create ((q', x') state formation).

Equation 2 defines the transition function $\delta \times \xi$ when the transition defines a new state for q and rest in the same state x ((q', x) state formation).

Equation 3 defines the transition function $\delta \times \xi$ when the transition rest in the same state q and defines a new state for x ((q, x') state formation).

A defended or forbidden transition is any transition resulting from a state (q, x) in synchronous composition such as there is an unknown controllable event where the transition is defined for G, the observed user behavior automaton and not defined for K, the specification automaton.

The supervisor synthesis according to Ramadge and Wonham should satisfy the active event constraint [13]:

$$(\Sigma\,(h\,(x) \cap \Sigma_0 \subseteq \Sigma\,(x)) \tag{4}$$

Where, h is the application which associates with any state x, of system to be commanded, a state q of the specification and such as x and q are accessible by the same event sequence from the start states and Σ_0 is the uncontrollable event set (events cannot be prevented from occurring).

4.2 Observed Behavior Analysis

In the absence of consideration of the uncontrollable events, we want to obtain supervisor's model which respects the requirements established in the specification.

We make use both algorithms of supervisor's synthesis ([13] and [14]) to define our algorithm for analyzing the observed behavior. This algorithm is based on algorithm for setting the defended commands.

The algorithm for setting the defended commands is shown in figure 2. The outputs of this algorithm are *Syn*, Synchronous composition construction between G and K, and t, defended transition. There are two inputs: K, the automaton specification, and G, the observed's user behavior automaton. It is based on two steps:

• **Synchronous composition construction between *G* and *K*:** the synchronous composition calculation is made according to the equations 1, 2 and 3.

• **Defended transition setting:** a transition (a command) which exists in the observed behavior model and which does not exist in the specification model is a defended transition.

Input: G and K
Output: *Syn* and t
1 **begin**
2 Build the synchronous composition, named Syn, between the observed user's behavior automaton (G) and the specification (K);
3 Setting the defended, transitions t, of the synchronous composition;
4 **end**

Figure 2. Algorithm for setting the defended commands

The synchronous composition gives three transitions types: the defended transitions, the authorized transitions that user do it and the authorized transitions that user don't do it. We notice that the defended transitions result from equation 2 of the synchronous composition. The authorized transitions that user don't do it result from equation 3.

Algorithm in figure 3 gives the user behavior type. It forbids any defended commands. It deletes in G the defended transitions and all the transitions resulting.

4.3 Anomaly Detection

In anomaly detection of user behavior, we need to distinguish between normal and intrusive behavior. So we analyze the

observed user behavior model and the specification model.

```
Input: G and t
Output: Behavior type
begin
    if G contains defended transitions then
        Forbid in G these defended transitions,
        Forbid in G all the transitions which
        can ensue from next states in the
        defended transitions ;
    else
        Authorized behavior
    end
end
```

Figure 3. Algorithm for analyzing the observed behavior

4.3.1 Cases analysis

The basic action of anomaly detection is to compare the observed user behavior model and the specification model. Two cases appear:

1) The observed user behavior model is included in the specification model thus the authorized model will be the observed's user model.

2) The observed user behavior model differs from the specification model. We differentiate the following three cases:

• If the specification automaton or the observed user behavior automaton is empty then the authorized model will be the empty automaton ($\varnothing, \Sigma, \varnothing, \varnothing, \varnothing$). (2.1)

• If the first transition is different than the authorized model will be an automaton having a single state. (2.2)

• If the states transitions from 1 to n are the same then the authorized model will be an automaton having the common transitions. (2.3)

The first case represents the normal behavior property. In fact every command typed by the user is an authorized command because it exists in the specification model.

The second case represents the intrusive behavior property. We differentiate every case:

Case (2.1) represents the malicious intrusive behavior property. In fact, the user behavior totally differs from the specification model.

Case (2.2) also represents the malicious intrusive behavior property. In fact, user behavior totally differs from the specification model.

The difference between cases (2.1) and (2.2) lies in their result. In the first case, the result, which is the authorized model, is an empty automaton and in the second case the result is an automaton having a single state. In practice, the result is the same.

Case (2.3) can represent the malicious intrusive behavior property or not. In fact, user behavior respects at the beginning the specification. Afterward, we have a change in this behavior. If this change is small (that is the different number of the commands from the model of specification is small as well as the nature of the used commands is offensive) then the behavior is intrusive but not malicious. In fact (that is the different number of the commands from the specification model is big as well as the dangerous nature of the used commands) then the behavior is intrusive and malicious.

After this cases differentiation, we are going to prove they obtained properties.

4.3.2 Behavior property

According to Ramadge and Wonham, we know that:

$$(\Sigma\,(h\,(x)) \cap \Sigma_0 \subseteq \Sigma\,(x)$$

and as we have no uncontrollable events ($\Sigma_0 = \emptyset$), this theorem becomes:

$$(\Sigma\,(h\,(x)) \subseteq \Sigma\,(x) \qquad (5)$$

Let's: $G = (Q, \Sigma, \delta, q_0, q_m)$ the observed's user behavior model, $K = (V, \Sigma, \xi, v_0, v_m)$ the specification model and $Syn = (X, \Sigma, \delta \times \xi, x_0, x_m)$ the synchronous composition.

Normal behavior property: The observed user behavior model is included in the specification model.

Let us show that in that case: the authorized model will be the observed user model. For this, it is enough to show that G verifies equation 5.

The synchronous composition will be: $G \cap K = G$ (because G is included in K).

$h : X \rightarrow Q$ and we have $X = Q$ Then $\forall x\ h\,(x) = q$.

So $\Sigma\,(h(x)) = \Sigma\,(x)$ Thus $\Sigma\,h(x) \subseteq \Sigma\,(x)$ is verified $\forall x$: thus the authorized model will be the observed's user model.

Intrusive behavior property: The observed user's behavior model differs from the specification model.

Malicious intrusive behavior property: If the specification automaton or the observed user behavior automaton is empty then the authorized model will be the empty automaton $(\emptyset, \Sigma, \emptyset, \emptyset, \emptyset)$.

If an automation is empty then we don't have any state: $G \cap K = \emptyset$ Thus $Syn = (\emptyset, \Sigma, \emptyset, \emptyset, \emptyset)$.

Malicious intrusive behavior property: If the first transition is different than the authorized model will be an automaton having a single state.

The first transition is different thus we have a blocking state so $Syn = (X, \Sigma, \emptyset, x_0, x_m)$.

Malicious or not intrusive behavior property: If the states transitions from 1 to '*n*' are the same then the authorized model will be an automaton having the common transitions.

Particular case of case (1)

5. Prototype for Linux/Unix Intrusion Detection

The prototype for Linux/Unix intrusion detection can survey a user and analyze his behavior.

5.1 Survey a user

There are two solutions to survey a user:

• The first solution consists in using the file .bash_history. But this file cannot give a strengthened and real-time history because when we use other shell, like csh,, this method cannot save the history. Either when you tape kill -9.

• The second solution is to develop a patch. It consists to modify file system in Linux, which are bashhist.c, histexpand.c, histfile.c, history.h and history.c. We do this because Linux is an open source (to obtain the patch e-mail: bentekaya.ines@voila.fr).

We can choose a user and we obtain the user's observed behavior. We can either choose a user and a day, shown in figure 4, and we obtain the user's observed behavior in this day. It is composed by time, process identifier (PID) and commands.

5.2 Analyse user behaviour

After survey a user, we can choose the files of the specification model and the observed user model, shown in figure 5.

The stages to follow for the specification analysis are: the choice of the specification model, the digitalization of the specification model, the generation of the specification automaton and the transformation of the specification automaton file to a .des file.

The stages to follow for the observed user behavior analysis are: the choice of the observed user behavior model, the digitalization

of the observed user behavior, the generation of the observed user behavior automaton and the transformation of the observed user behavior automaton file to a .des file.

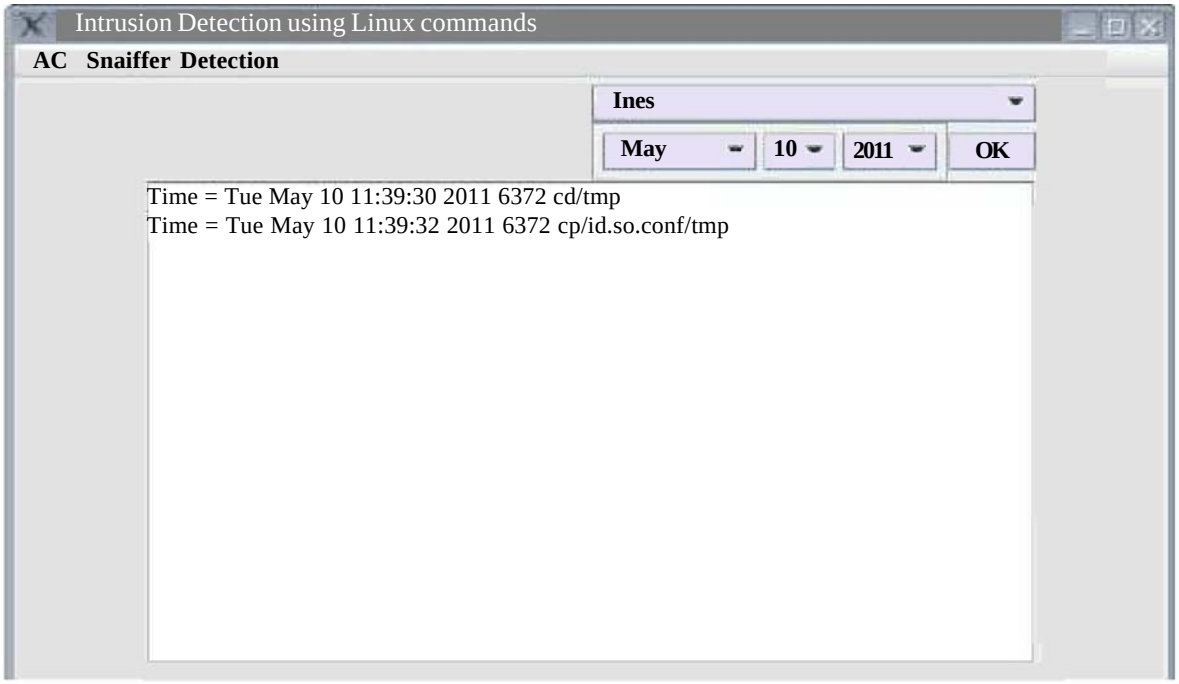

Figure 4. User's observed behaviour in a chosen day

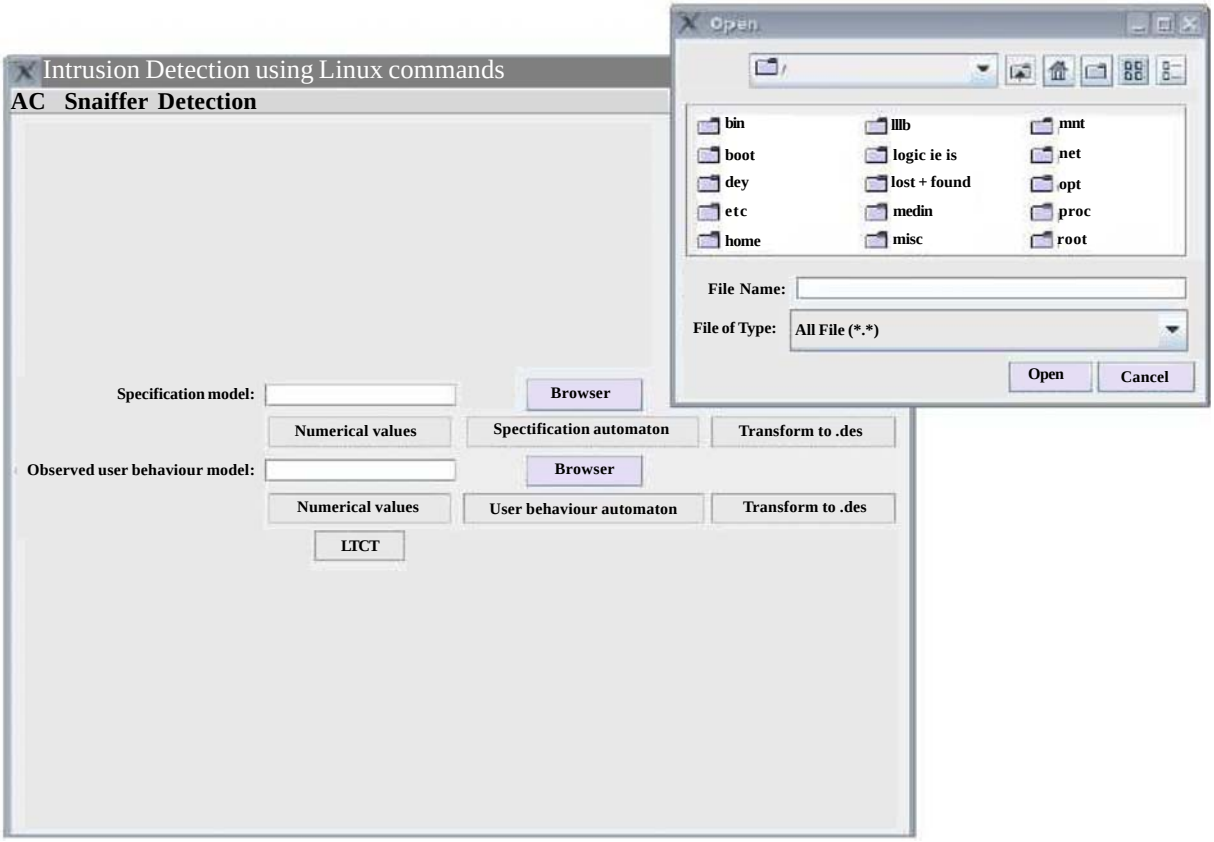

Figure 5. Choice of the models

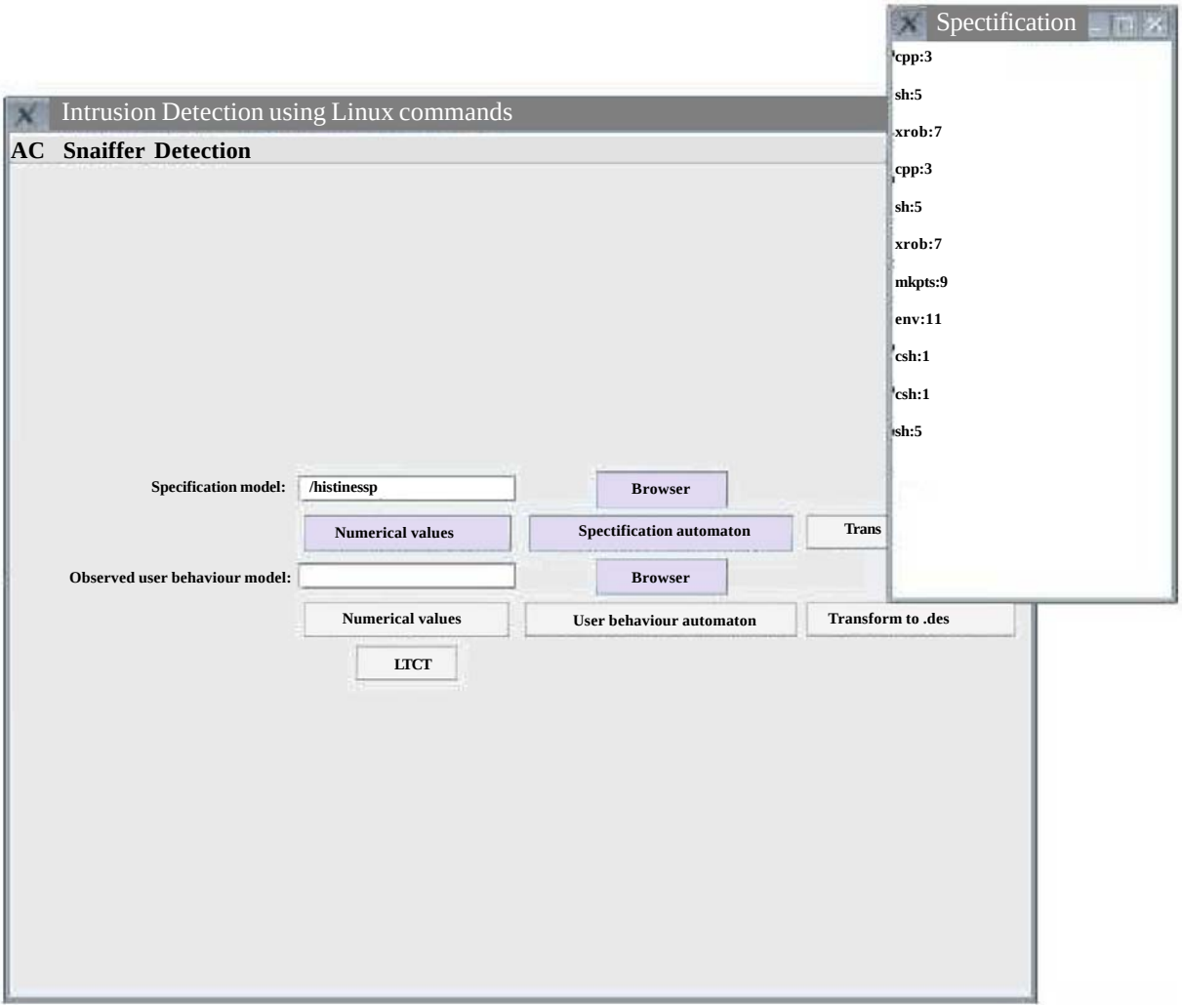

Figure 6. Numerical values of the commandsin the specification model

Afterward, we build the synchronous composition. We apply our algorithm for setting the defended commands then our algorithm for analyzing the observed behavior.

All of theses stages will be explain in the followings paragraphs.

• **Choice of the specification model:** The specification model is stored in a file. At the beginning, we have only browser buttons enabled in figure 5. They serve to choose the specification model and the observed's user behavior model. When we have chooses the file corresponding to the specification model, the numerical values button will be enabled.

• **Digitalization of the specification model:** This step is important, because when we use the LTCT tool [15], commands should be numerical. When we click the numerical values button, we obtain a new window containing two columns: the first one is the commands names and the second one is the numerical values of the corresponding commands. For example in figure 6, the numerical value are: cpp : 3 ; sh : 5 ; xrdb : 7 ; mkpts : 9 ; env : 11 et csh : 1.

• **Generation of the specification automaton:** Once the commands are digitized, the specification automaton button becomes enabled. This button allows giving states and transitions of the specification automaton as shown in figure 7.

• **Transformation of the specification automaton file to a.des file:** The button transform.des transforms the file containing the specification automaton (named spuser.ads). We use the FD operation of LTCT procedures from figure 8. The figure 9 displays an example of spuser.des file.

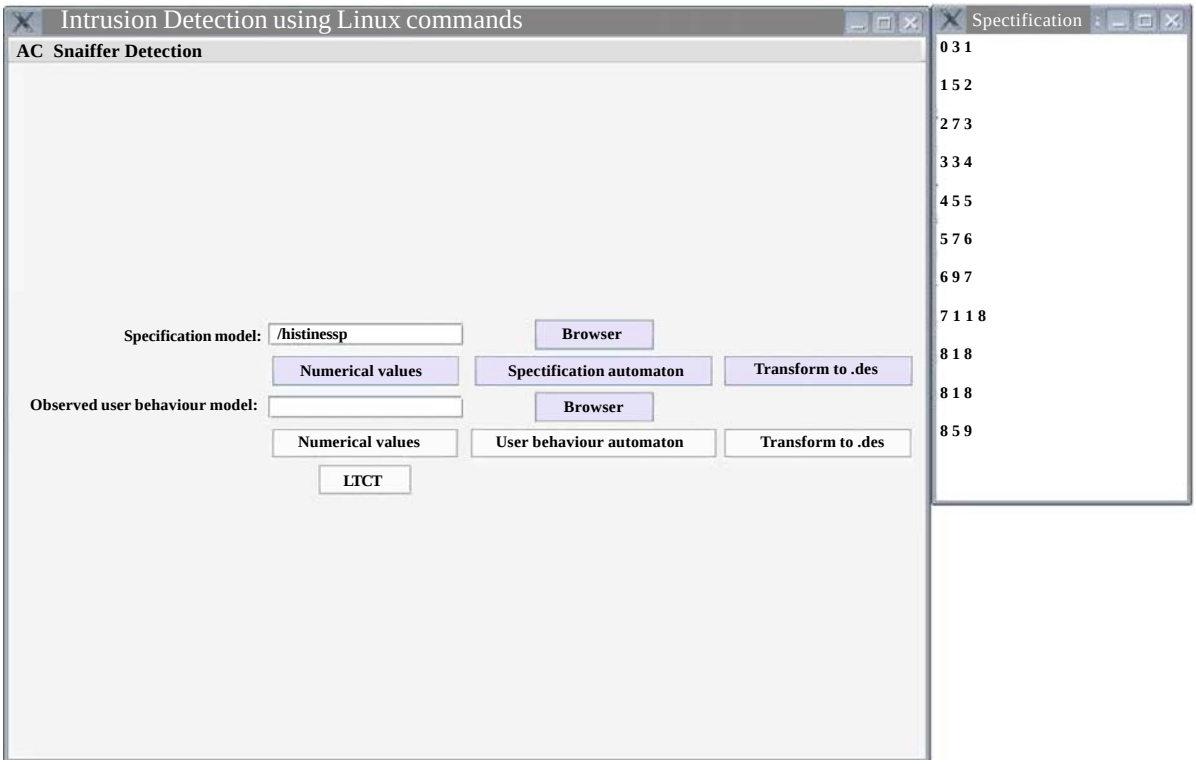

Figure 7. Specification automaton

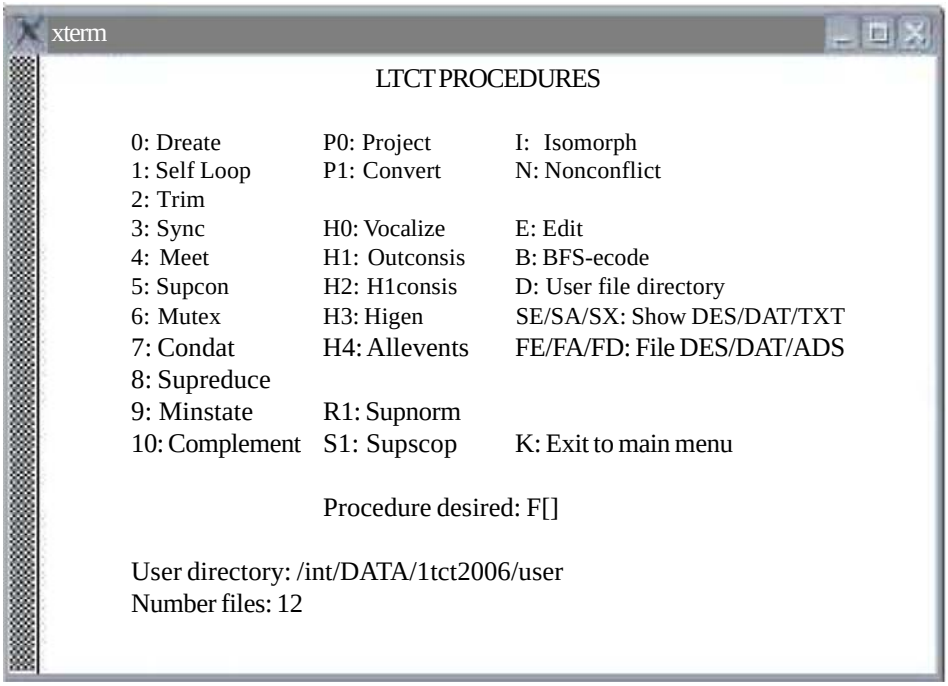

Figure 8. LTCT Procedures

• **Choice of the observed's user behavior model:** the first step consists in the choice of the observed's user behavior model file.

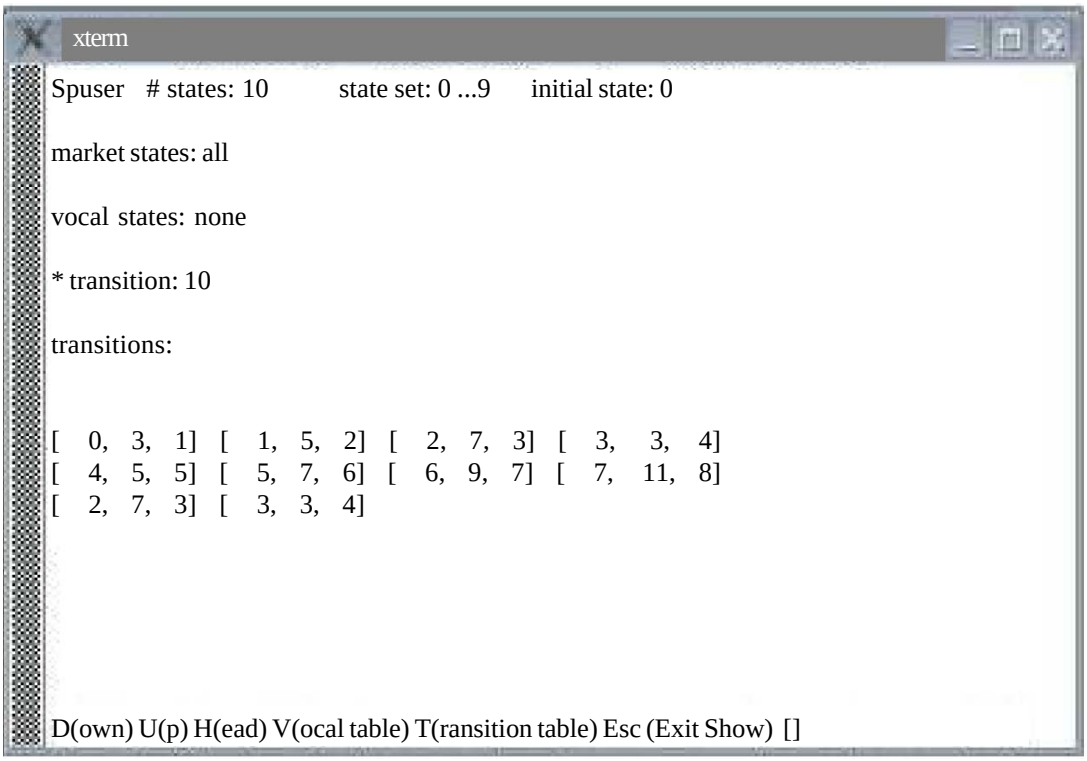

Figure 9. Contents of spuser.des file

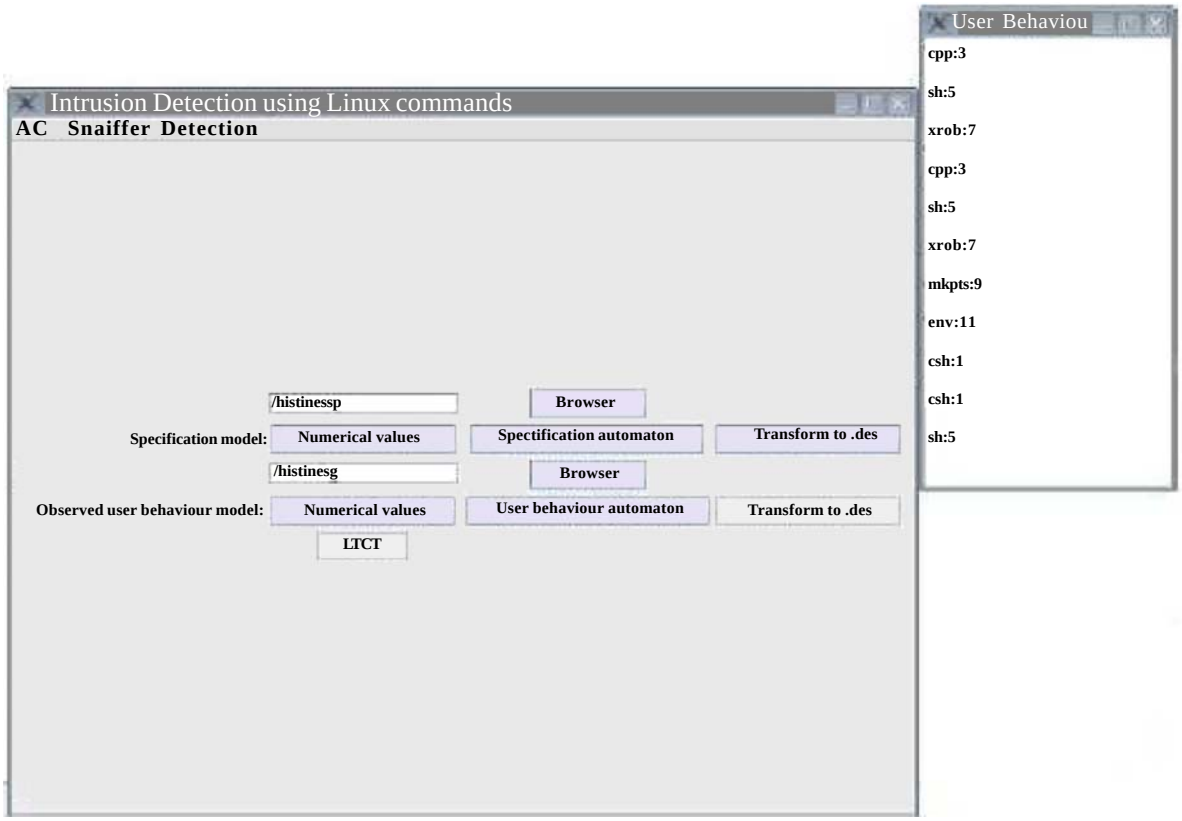

Figure 10. Numerical values of the commandsin the observed user behavior

• **Digitalization of the observed's user behavior model:** This step is important, because when we use the LTCT tool, commands should be numerical. When we click the numerical values button, we obtain a new window containing two columns: the first one is the commands names and the second one is the numerical values of the corresponding commands. For example, the numerical value in figure 10 are: cpp : 3 ; sh :5 ; xrdb : 7 ; mkpts : 9 ; env : 11 et csh : 1.

• **Generation of the observed's user behavior automaton:** Once the commands are digitized, the observed user = behavior automaton button becomes enabled. This button allows giving states and transitions of the observed user behavior automaton as shown in figure 11.

• **Transformation of the observed's user behavior automaton file to a .des file:** The button transform.des transforms the file containing the observed user behavior automaton (named guser.ads). We use the FD operation of LTCT procedures to do that. The figure 12 displays an example of guser.des file.

• **Build the synchronous composition:** When we obtain both .des files relative to the specification model and to the observed's user behavior, the button LTCT becomes enabled. We can so build the synchronous composition using the procedure number 5 of figure 13. The obtained result is an automaton formed by the transitions of figure 14.

• **Application of algorithm for setting the defended commands:** We find that the transition 7 connecting the state 9 to 10 is defended because it is not defined in the specification model.

• **Application of algorithm for analyzing the observed behavior:** The behavior of this user is authorized until it uses the xrdb command. So our supervisor forbids this behavior.

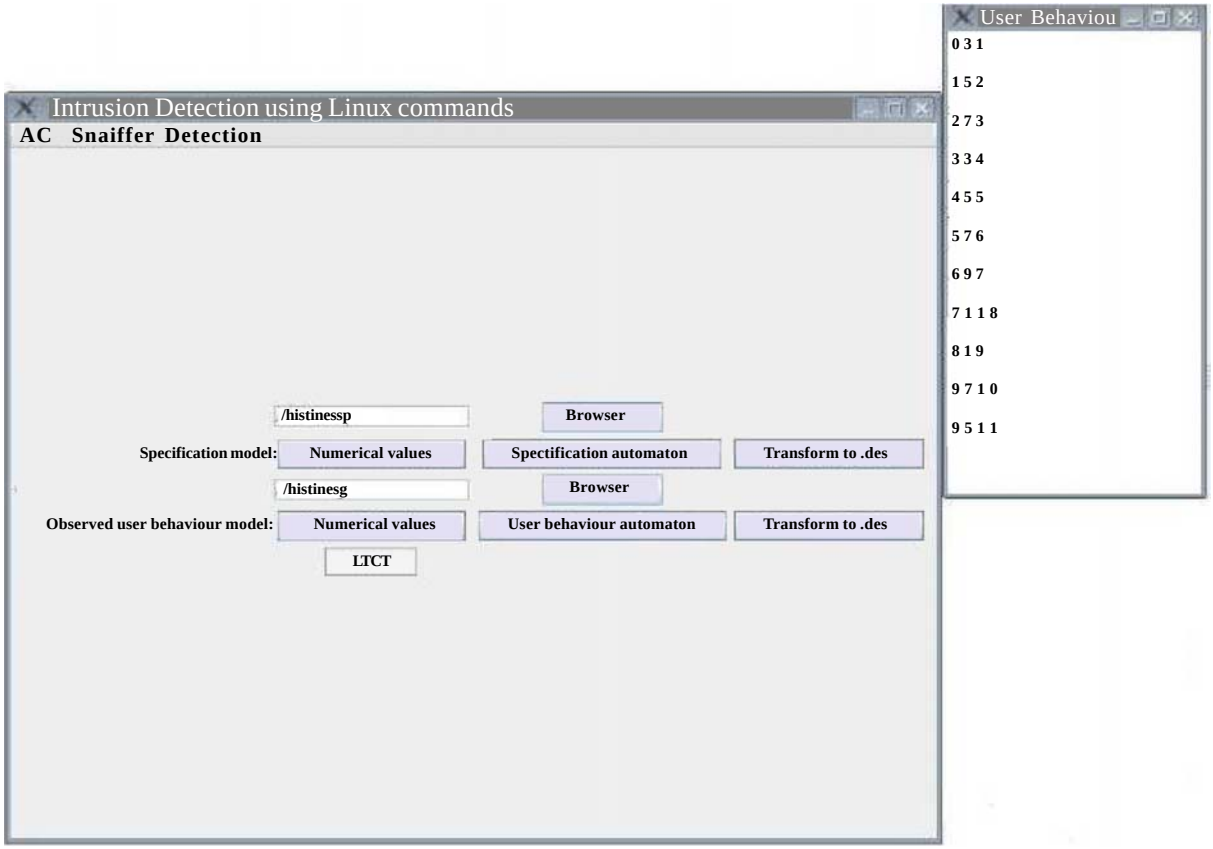

Figure 11. Observed user behavior automaton

6. Conclusion

In this paper, we have proposed a new method for anomaly detection of user behavior. It uses the specifications formalization

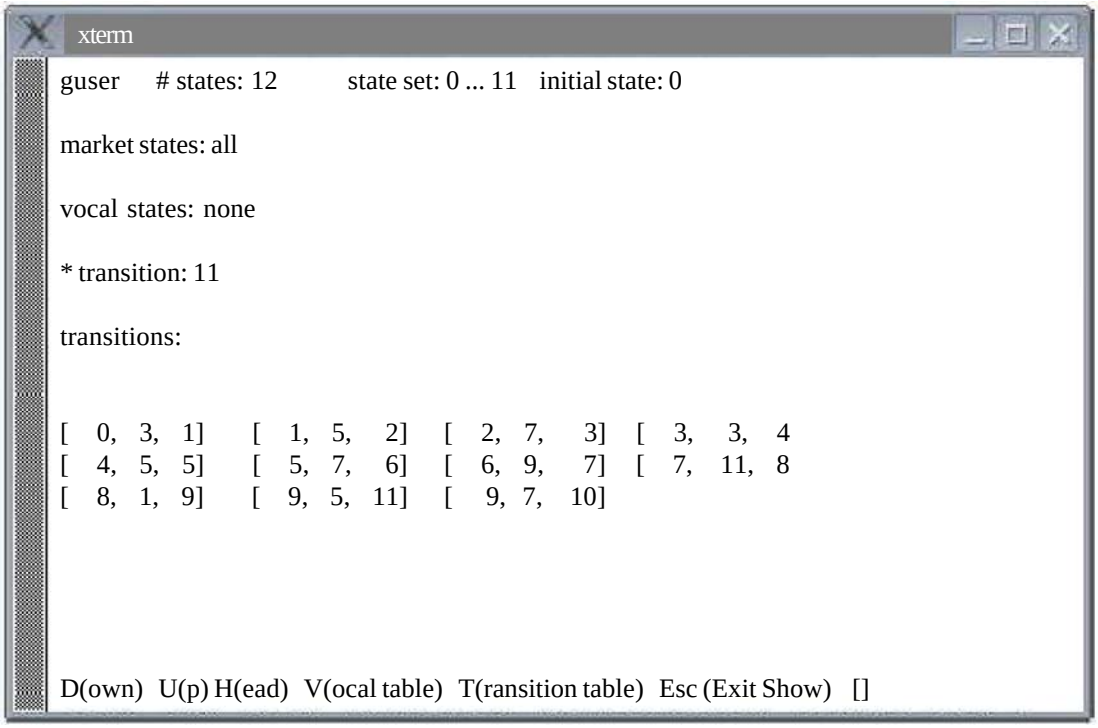

Figure 12. Contents of guser.des file

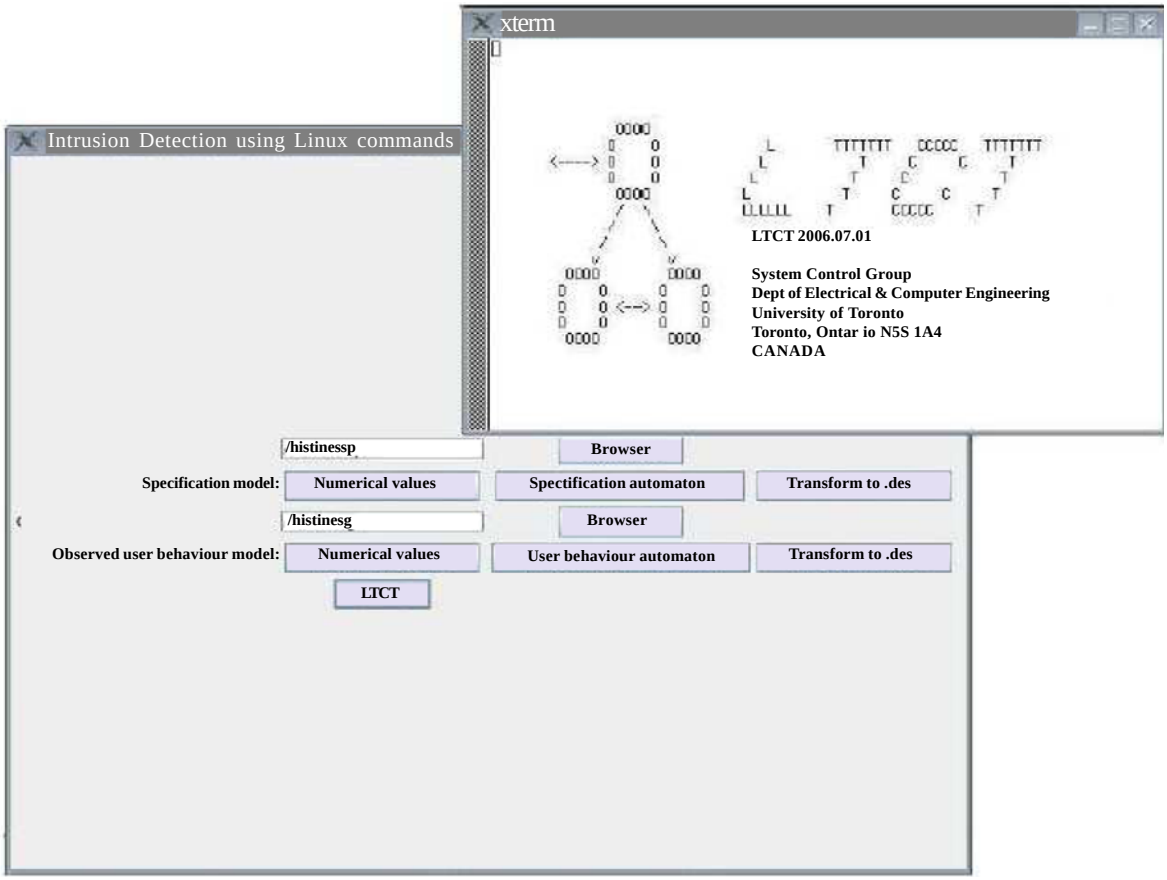

Figure 13. LTCT button

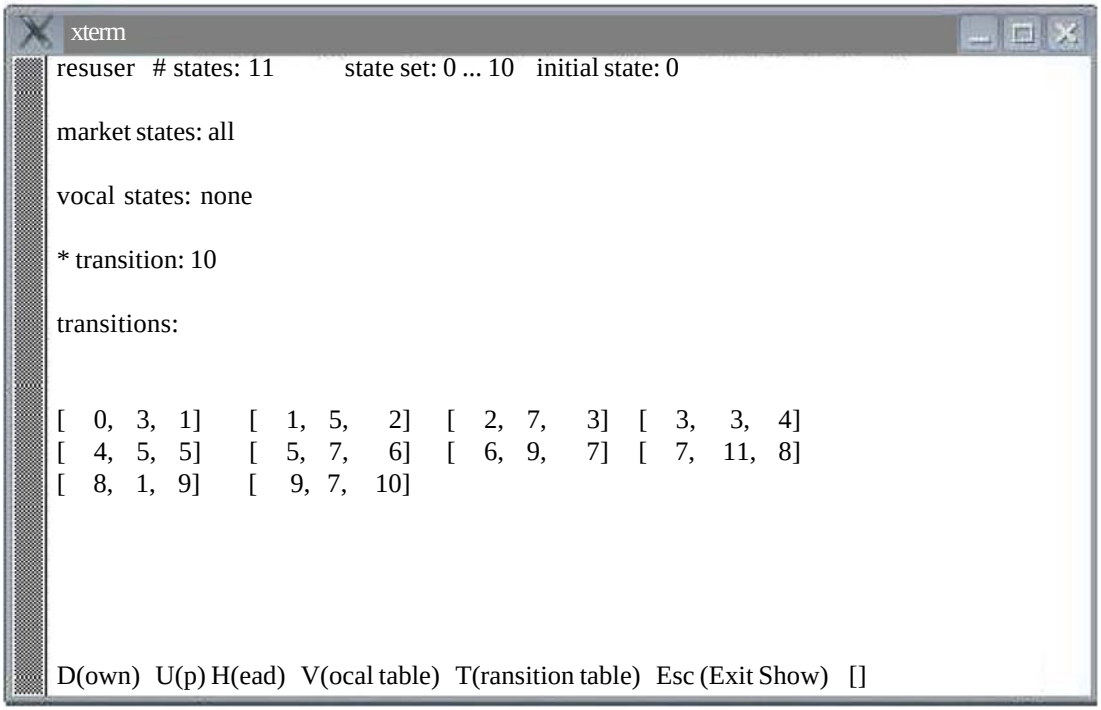

Figure 14. Synchronous composition

and the observed user behavior. Two advantages characterize our proposed algorithm for analyzing the observed behavior. The first advantage is that the algorithm result is a structure. The second advantage is the way of searching faults or intrusions.

This method is applied to distinct normal user behavior from intruders' behavior. It has lead to the prototype for Linux/Unix intrusion detection tool development.

In future work, we plan to improve our algorithm of observed behavior analysis to take into account commands parameters.

References

[1] Anderson, J. P. (1980). Computer Security Threat Monitoring and Surveillance, *Technical report*, Washing, PA, James P. Anderson Co..

[2] Powell, D., Stroud, R. (2003). Conceptual Model and Architecture of MAFTIA, Eds., MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) project deliverable D21, *LAAS-CNRS Report* 03011.

[3] Matheï, C. (2004). Ouverture des réseaux IP d'entreprise : risques ou opportunité ? [Online]. Available: http://www.awt.be/contenu/tel/res/IPforum23-04_Réseau unifié et sécurisé.pdf.

[4] Cloete, B. E., Venter, L. M. (2001). A comparison of Intrusion Detection systems, *Computers & Security*, 20 (8) 676-683, Dec.

[5] Patrizio, A. (2006). Linux Malware On The Rise. [Online]. Available: http://www.internetnews.com/devnews/article.php/3601946.

[6] Santana, M. (2009). Chapter 6 - Linux and Unix Security, Computer and Information Security, Handbook, p. 79-92.

[7] Koral Ilgun , Richard A. Kemmerer, Phillip A. Porras. (1995). State Transition Analysis: A Rule-Based Intrusion Detection Approach. *Journal IEEE TRANSACTIONS on Software Engineering*, 21 (3) 181-199.

[8] Ilgun, K. (1992). USTAT - A Real-time Intrusion Detection System for UNIX, Master's Thesis, University of California at Santa Barbara, Nov.

[9] Schonlau, M., DuMouchel, W., Ju, W. H., Karr, A. F., Theus, M., Vardi, Y. (2001). Computer Intrusion: Detecting Masquerades, *Statistical Science*, 16 (1) 1–17.

[10] Lane, T., Brodley, C E. (1997). Sequence matching and learning in anomaly detection for computer security. *In*: AAAI Workshop: *AI Approaches to Fraud Detection and Risk Management*, p. 43–49. AAAI Press.

[11] Theus, M., Schonlau, M. (1998). Intrusion detection based on structural zeroes. *Statistical Computing and Graphics Newsletter*, 9, p. 12–17.

[12] Roy, M. (2003). Masquerade detection using enriched command lines. *In*: Proceedings of international conference on Dependable Systems and Networks (DSN-03), p. 5-14, June.

[13] Ramadge, P. J., Wonham, W. M. (1987). On the Supremal Controllable Sublanguage of a Given Language, *SIAM J. Control and Optimization*, 25 (3) 637-659.

[14] Kumar, R., Garg, V., Marcus, S. I. (1991). On controllability and normality of discrete event dynamical systems, *Systems and Control Letters*, 17, p.157-168.

[15] Wonham, W. M. (2000). Supervisory control of discrete-event systems: an introduction. *In*: Industrial Technology, Proceedings of IEEE International Conference, 1, p. 474 - 479.