Coloured Petri Nets for Modeling of Host-Based Attacks

Eduardo Aguirre, Karen A. García-Gamboa, and Raúl Monroy

Tecnológico de Monterrey, Campus Estado de México
Carretera a Lago de Guadalupe Km. 3.5, Atizapán, Estado de México, 52926, Mexico
{a00467871,kazurim,raulm}@itesm.mx

Abstract. Intrusion Detection Systems (IDSs) often use an attack characteristic signature in order to detect the presence of an attack during a computer session. The creation of one such an attack signature is a hard, error-prone task and requires a great deal of expertise both in computer security and in network operating systems. To compound the problem, the number of signatures in an IDS database grows up rapidly and so does the detection time. This is because, for it to be able to detect an attack, an IDS must have the corresponding attack signature, even though if the attack is a variation of other one, so called a *mimicry attack*. Therefore, there exists the need of creating an attack model that is both easy to understand and amenable for the automatic application of an IDS to readily approach attack detection. This paper proposes the use of Coloured Petri Nets (CPN's) to model computer attacks. In particular, we will apply CPN's to model user-to-root host-based attacks. We will see that a CPN-based attack model can be readily used by an IDS to detect the attack and some of its variations. The CPN model is easy to build and to understand. CPN model building is not time consuming. What is more, the construction of a CPN-based attack model can be automated, albeit this issue will be largely ignored here, as it goes beyond the purpose of this paper.

1 Introduction

From the beginning of the Internet, there has been a constant war between people who want to steal information or cause some other kind of damage to computer users and people who want to maintain the (incipient) law and order over the Internet. Computer attacks have already resulted in countless losses. Lost assets range from personal, private information, to money and image or reputation.

Intrusion detection is concerned with the timely discovery of any activity that jeopardises the integrity, availability or the confidentiality of an IT system. There are two types of Intrusion Detection Systems (IDSs). A misuse IDS aims at detecting a known pattern of malicious computer usage or network traffic. An anomaly IDS aims at detecting any deviation to ordinary, expected behaviour. Mostly, former IDSs were of type misuse.

This paper is concerned with misuse intrusion detection. Misuse IDSs often use the characteristic signature of an attack in order to detect the presence of the attack during a computer session. The creation of one such an attack signature is a hard, error-prone task and requires a great deal of expertise both in computer security and in network operating systems. To compound the problem, the number of signatures in an IDS database grows up rapidly and so does the detection time. This is because, for it to be able to detect an attack, an IDS must have the corresponding attack signature, even though if the attack is a variation of other, already in the IDS attack database (so called a *mimicry attack*). Thus, there exists the need of creating an attack model that is both easy to understand and amenable for the automatic application of an IDS to readily approach attack detection.

This paper suggests the use of Coloured Petri Nets (CPN's) to model computer attacks. In particular, it shows how to apply CPN's to model user-to-root host-based attacks. We shall see that a CPN-based attack model can be readily used by an IDS to detect the attack and some of its variations. One of the key advantages of using CPN's to model computer attacks is that the model is general enough to capture a number of attacks and yet sufficiently particular to speak out the more relevant details of the attack that readily lead to its detection. We will argue that the CPN model is easy to build and to understand and that CPN model building is not time consuming. What is more, the construction of a CPN-based attack model can be automated, albeit this issue goes beyond the purpose of this paper and so will be largely ignored here.

This paper introduces a methodology for the modelling of an attack using the system calls generated during attack execution. The paper argues the advantages of modelling attacks using CPN's against of using standard signatures.

Paper Overview The rest of this paper is organised as follows. In §2 we present a little overview of the modelling methodology. §3 and §4 explain what is a PN and a CPN. In the section §5 we present our modeling technique along with the methodology and the attacks used in our research, in §6 a brief comparison with other similar methodologies is presented, finally we conclude the paper with the section §7 and we give directions for further work.

2 Approach

A CPN is a graphic model capable of representing an action sequence that is the cause of an event. We can use this model in a general (abstract out details by collecting into a single, high-level event a number of low-level system calls) or specific (system calls) way to get a better knowledge of the event. Because of these characteristics we use CPN's to model host based attacks. The main purpose of doing this is to do a better representation of the attack, and a model that can be used by an IDS.

Because of the nature of a CPN, it is possible to detect some variations of an attack through the model. To create an accurate model it is necessary to consider the following points:

1. The model must contain only the needed steps to accomplish the attack.
2. Every way in which those steps can be done must be in the model.
3. The transitions in the CPN can be activated in one way or another. The execution of some actions can be interchanged.

To use a CPN as a model we must create a methodology based on the previous points.

3 Petri Nets (PN's)

PN's were created in August 1939 by Carl Adam Petri for the purpose of describing chemical processes. A PN is a mathematical modeling language. It consists of places, transitions, and arcs that connect them. Input arcs connect places with transitions. Output arcs start at a transition and end at a place. Places can have tokens; the current state of the modeled system is given by the number and type of tokens in every place.

PN's model activities using places and transitions. A place represents a system in a time n, a transition connects two or more places. Transitions are only allowed to fire if they are enabled, which means that all the preconditions for the activity must be fulfilled. The preconditions ensure thet there are enough tokens available in the input places. When the transition fires, it removes tokens from its input places and adds some into the output places. The number of tokens removed or added depends on the cardinality of each arc. The interactive firing of transitions in subsequent markings is called the token game.

PN's are good enough for describing and studying systems that are characterised as being concurrent, asynchronous, distributed, parallel, nondeterministic, and stochastic. Since PN's are a graphical tool, they can be used as a visual-communication aid similar to flow charts, block diagrams, or networks. Moreover, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. In a PN is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems.

4 Coloured Petri Nets (CPN's)

A CPN is an extension of PN. So, a CPN is a graphical oriented language for the design, specification, simulation and verification of systems [1]. The principal reason for the success of these kinds of net models are the fact that they have a graphical representation and a well-defined semantics allowing formal analysis. It is in particular well-suited for systems that consists of a number of processes which communicate one another. Principal examples of CPN's application areas are communication protocols, distributed systems, automated production systems and work flow.

A CPN consists of three different parts: the *net structure* (i.e., the places, transitions and arcs), the *declarations* and the *net inscriptions* (i.e., the various text strings which are attached to the elements of the net structure). Transitions

have two kinds of net inscriptions: names and guards. Arcs only have one kind of inscription: the arc expressions. All net inscriptions are positioned next to the corresponding net element and to make it easy to distinguish between them we write names in plain text, colour sets in italics, while initialization expressions are underlined and guards are enclosed in square brackets. A CPN may have several other kinds of inscriptions (e.g., describing hierarchical relationships and time delays).

Each CPN has a set of declarations, which we position by convention in a box with dashed lines. The declarations introduce a number of colour sets, functions, operations, variables and constants, which can be used in the net inscriptions of the corresponding CPN, particularly in the guards, arc expressions and initialization expressions. The declarations of a CPN can be made in many different languages, e.g., by means of standard mathematical notation or by means of many sorted sigma algebras. Each colour set declaration introduces a new colour set, whose elements are called colours. Every colour set declaration implicitly declares a set of constants (the colours of the colour set). Moreover, the colour set declaration can implicitly declare some standard functions and operations which can be used on the colours of the colour set. A declared colour set can be used [1]:

- In the declaration of another colour sets.
- In the declaration of variables (having the colour set as type).
- In the declaration of functions, operations and constants (e.g., a function may map from one colour set into another colour set).
- In the colour set inscription of a place (indicating that all tokens on the place must have token colours which belong to the colour set).

5 Modeling Host-based Attacks with CPN

In this section we will explain the methodology for modeling attacks throguh CP-nets. The models we have created are based on the system calls [2] produced by the execution of an attack. It is important to say that the model abstraction level can be changed to make it more understandable.

To create a model we analize the system calls generated by an attack execution.

We modeled five different attacks, this is showed in 1.

We logged the execution of each one of these attacks through the *strace* utility:

$$strace - f - F - t - T - oOutputFileExploitToMonitor \tag{1}$$

The output of the previous command is a log with all the system calls generated by the exploit, these system calls are the base of our methodology. The preconditions for the transition to be activated are characteristics of the token or the system, the postconditions are changes in the token or the system consequence of the transition occurrence.

Table 1. Modeled Attacks

Attack	OS
vmsplice (1)	Fedora 8
vmsplice(2)	Fedora 8
execve/ptrace	Red Hat 7
kon	Red Hat 9
do-brk vma	Red Hat 9

Transitions must be created dynamically according to some set of predefined elements. Each of these transitions must have places before and after it, the colour depends on the elements of the transition.

Because of the nature of the problem it is not possible to pre-define the elements of the transitions, Following [3], we have divided the transitions into two principal elements: conditions $C = \{c_1, c_2, ..., c_n\}$ and consequences $O = \{o_1, o_2, ..., o_n\}$.

Conditions are observable characteristics of a system, which we can predefine and use to generate actions that represent a condition change. However for each condition we need to define a threshold: just one change in any condition causes an action and a new set of conditions. Formalizing: Let T be a set of Thresholds $T = \{t_1, t_2, \ldots, t_n\}$. Let $c_i \epsilon C$ and $t_i \epsilon T$, if c_i and t_i are such that the features of c_i match the threshold t_i. Then we create a new transition such that it has a previous state with color C and a posterior state with color C' where $C' \neq C$ *and* $C' = O$ such that O is the set of consequences or postconditions of the transition and C is the set of preconditions of the transition.

The a priori definition of conditions and thresholds makes possible to dynamically create actions when changes in the conditions occur. This allow us to create transitions in a CP-net, the initial set C are the preconditions, O the postconditions, and T represents the transitions between these sets.

The steps for modelling an attack are the following:

1. Create a log of the system calls generated by the execution of an attack.
2. Look sequentially into the log for an occurrence of any of the conditions and thresholds defined a priori.
3. If a condition change is such that the features of the condition match the threshold, look for the values of the condition in the system before the threshold fires and create an state or merge with another existent, which color and values depends on the condition before the ocurrence of the threshold.
4. Create a transition which color and values depends on the threshold found.
5. Look for the values of the condition in the system after the threshold was fired and create an state or merge with another existent, which color and values depends on the condition after the ocurrence of the threshold.
6. Go back to step 2 until there is no match of any condition with a threshold.

Figures 1 and 2 are the definitions of colors and threshold values used in our experimentation. These data are the base used for modelling, if a threshold

occurs in the log, a transition with its input and output states is created, in this way the CPN is created while the log is being analysed.

The proposed methodology allows to detect modified attacks that insert no operations, however to perform mimicry detection some modification to the methodology must be made to detect syscalls substitution and interchange.

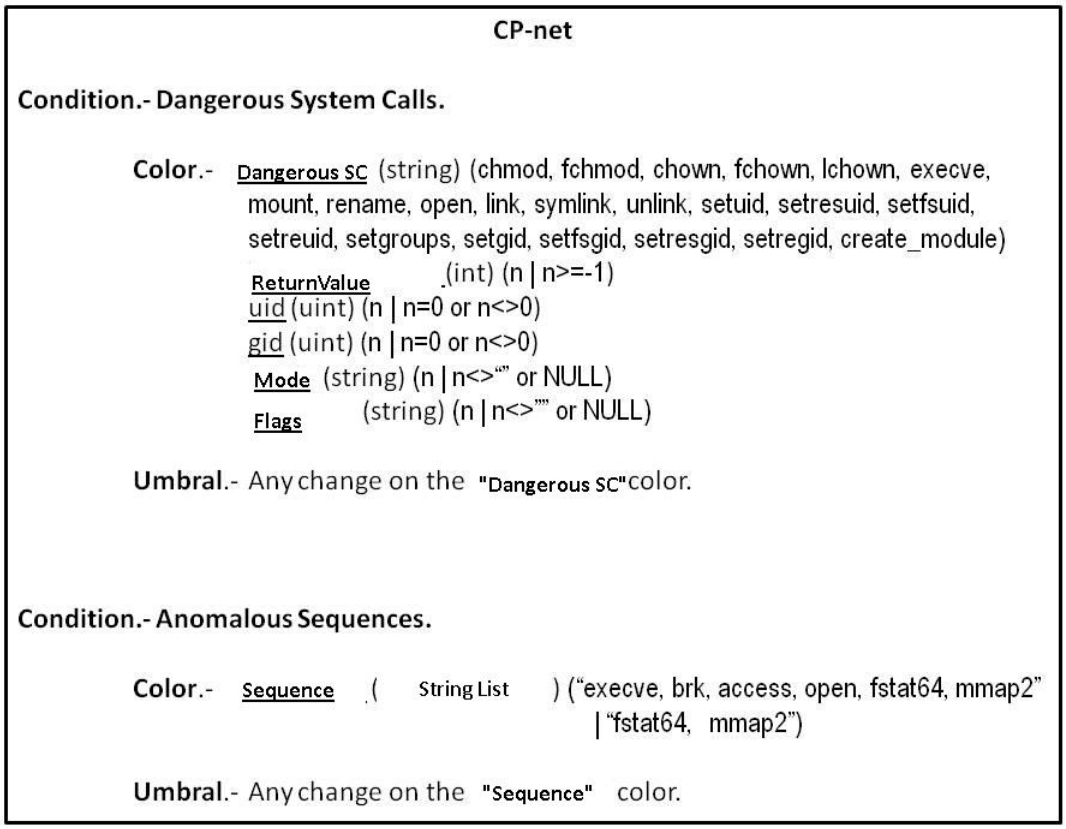

Fig. 1. Colours 1 and 2

Figures 3 and 4 show two CPN's created by following our methodology. Due to paper size constraints it was not possible to show a figure that shows the creation of a CPN step by step.For more details, contact the first author at ironwk85@gmail.com.

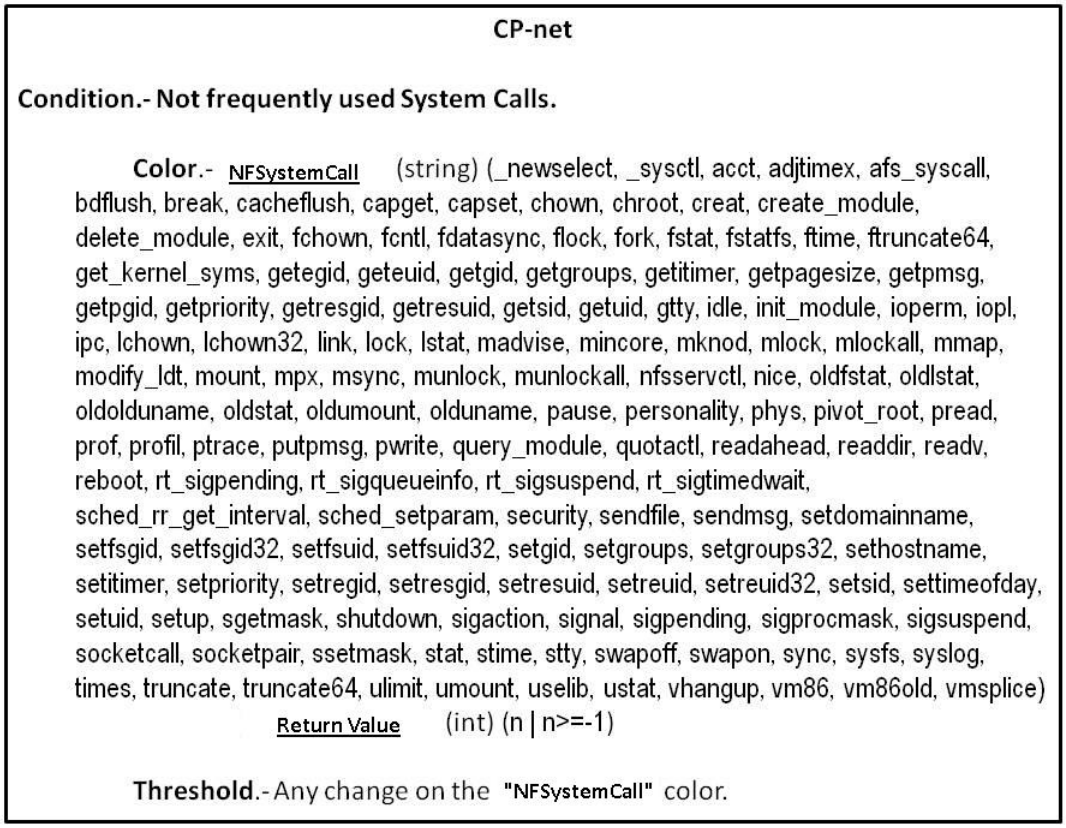

Fig. 2. Colour 3

6 Related Work

There are two similar approaches that try to create an attack model similar to a CPN. One is [4] that proposes to create an attack model using a language called EDL (Event Description Language). This model is similar but is not as popular as a CPN. The created model is easier and faster to create than a common signature but it does not have the abstraction characteristic that a CPN has. In that paper the elements modelled are events. We think that there is no need to create a new language based on CPN's, we create models using CPN's and system calls not events, the result are a model that can be abstracted, capable of detect some modifications of an attack and with a semi automatic creation, all of this without learn a new language.

The second approach uses hierarchical CPN's [5]. It presents a very good approach to modelling network based attacks. They use events for the modelling

but do not have any intention to automate their models. Our job only focuses on host based attacks and tries to create a model that can be automated, and capable of detecting attacks and some modifications of them using system calls.

7 Conclusions

CPN's are a good approach to model host based attacks because of the characteristics of the method, however a good methodology is necessary to take advantage of them. The methodology we present must be modified to allow mimicry attack detection. However the generated models seem to be a good representation of the attack.

The methodology presented is also able to be automated, being this some further work to do, along with the isolation of the attack, that is commonly contained into a huge log of system calls.

The model can apparently be the input of an IDS. More experiments and more analysis are required, but the initial tests show a positive result. This was done by the execution of the attack several times and using the obtained model to check if the attack could be detected at some point. Successful results were obtained in all the tests. When some modifications were done to the attack, just the insertion of no operations allow the model to detect the attack, other modifications make the model to do not recognize the attack.

References

1. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use. Springer (1997)
2. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. Journal of Computer Security **6**(3) (1998) 151–180
3. Girault, C., Valk, R.: Petri nets for systems engineering: a guide to modeling, verification, and applications. Springer Verlag (2003)
4. Meier, M., Schmerl, S., König, H.: Improving the efficiency of misuse detection. In: DIMVA. (2005) 188–205
5. Wu, R., Li, W., Huang, H.: An attack modeling based on hierarchical colored petri nets. Computer and Electrical Engineering, International Conference on **0** (2008) 918–921

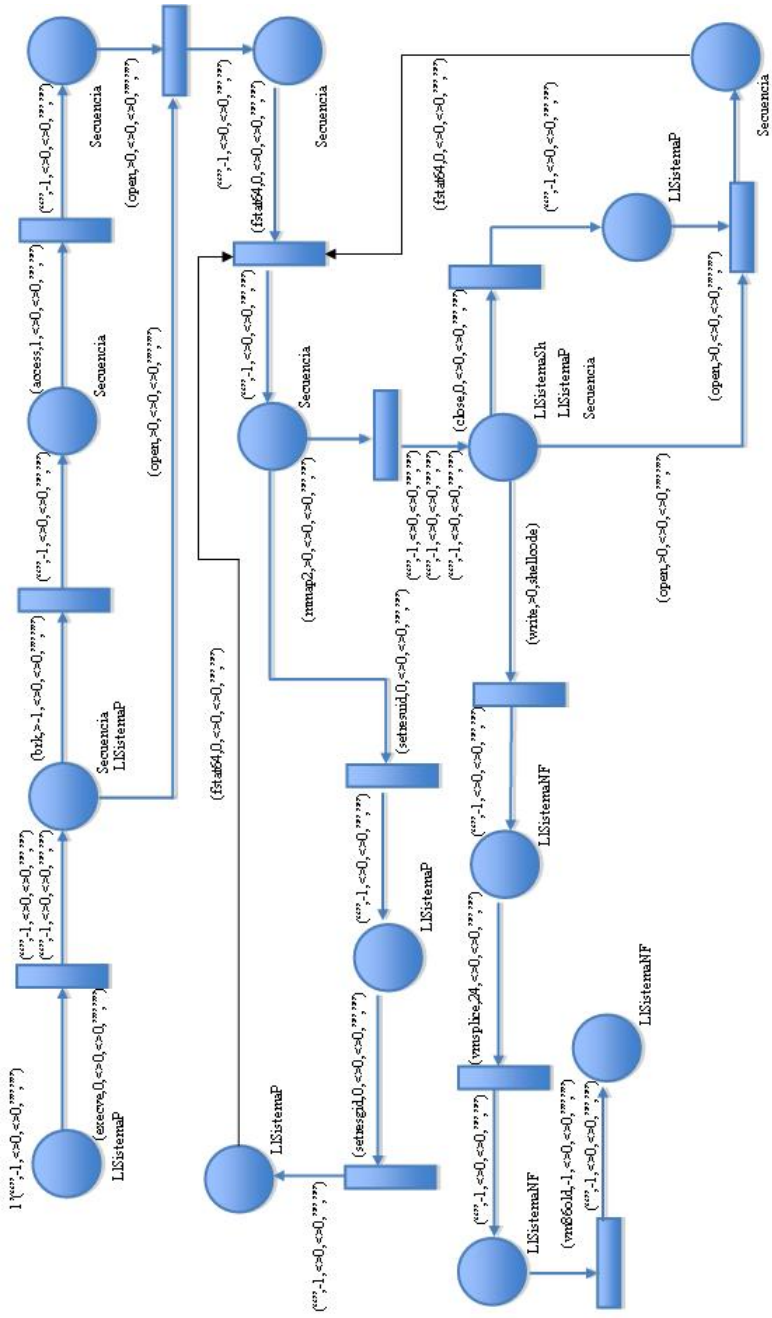

Fig. 3. Example 1: Attack Model

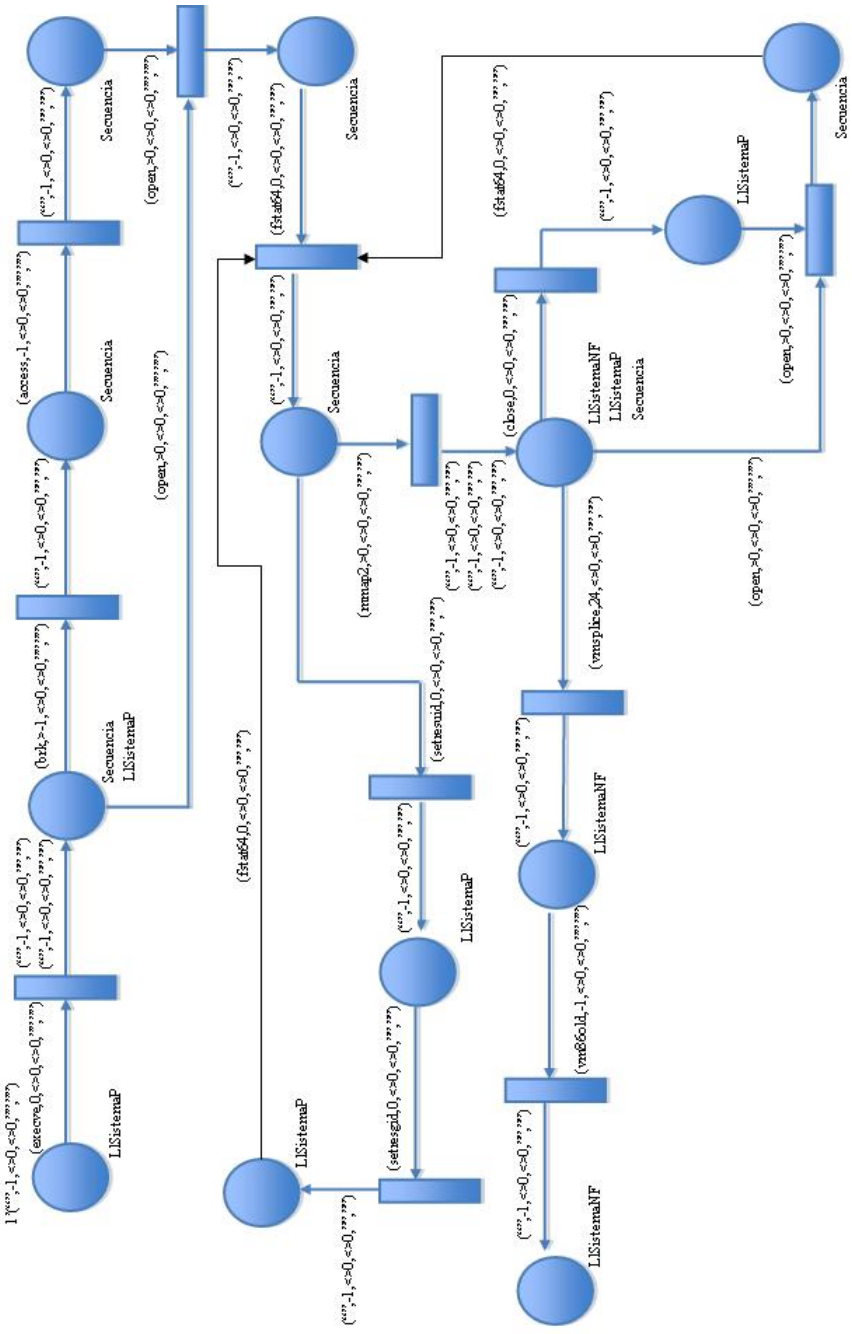

Fig. 4. Example 2: Attack Model