Int. J. Communications, Network and System Sciences, **2009, 2, 857-873**
doi:10.4236/ijcns.2009.29100 Published Online December 2009 (http://www.SciRP.org/journal/ijcns/).

Forensic Investigation in Communication Networks Using Incomplete Digital Evidences

Slim REKHIS, Jihene KRICHENE, Noureddine BOUDRIGA
CN&S Research Lab, University of the 7th of November, Carthage, Tunisia
E-mail: *slim.rekhis@isetcom.rnu.tn*, *jkrichene@gmail.com*, *nab@supcom.rnu.tn*
Received July 16, 2009; *revised September* 10, 2009; *accepted October* 21, 2009

Abstract

Security incidents targeting information systems have become more complex and sophisticated, and intruders might evade responsibility due to the lack of evidence to convict them. In this paper, we develop a system for Digital Forensic in Networking, called DigForNet, which is useful to analyze security incidents and explain the steps taken by the attackers. DigForNet combines intrusion response team knowledge with formal tools to identify the attack scenarios that have occurred and show how the system behaves for every step in the scenario. The attack scenarios construction is automated and the hypothetical concept is introduced within DigForNet to alleviate missing data related to evidences or investigator knowledge. DigForNet system supports the investigation of attack scenarios that integrate anti-investigation attacks. To exemplify the proposal, a case study is proposed.

Keywords: Formal Digital Investigation, Incident Response Probabilistic Cognitive Map, DigForNet, Anti-Forensic Attacks Investigation, Attack Scenarios Reconstruction

1. Introduction

Considering the state of digital security incidents which has dramatically increased in terms of complexity, number, and sophistication, it becomes evident that the traditional ways of protecting information systems (e.g., Firewalls, IDSs) are no longer sufficient. Faced to this situation, security experts have started giving a great interest to a novel discipline called the digital investigation of security incidents, which is defined by the literature as preservation, identification, extraction, documentation and interpretation of computer data [1]. Digital investigation aims to perform a post-incident examination on the compromised system while achieving several objectives including evidence collection, attack scenarios construction, and results argumentation with non refutable proofs.

Performing a digital investigation is a very complex task for many reasons. First, attacks may use multiple sources and become difficult to trace using the available trace-back techniques. Second, systems may not be initially prepared for investigation, leading to the absence of effective logs or alerts to be used during the analysis of the incident. Third, the attackers may use a number of techniques to obfuscate their location or to hide traces on the system that could be used to prove their malice. Fourth, attack scenarios may use several automated tools that create intensive damaging activities on the compromised systems. A large amount of data should thus be analyzed and several evidences need to be identified and extracted.

To face the above complexity, the digital investigation should, first, be well structured by reconciling both the expertise of the incident response team (IRT) and the use of formal reasoning techniques about security incidents. This reconciliation allows to: 1) better filter the data to be analyzed and source of evidences to be explored, based on the skills developed by the IRT; 2) validate the results of the formal techniques, by the IRT, before presenting them and also use them to improve the content of the attacks library. Second, digital investigation should integrate the use of formal techniques that are useful to develop non-refutable results and proofs, and avoid errors that could be introduced by manual interpretations. Moreover, it should consider the development of tools to automate the proof provided by these formal methods. Third, since the collected evidences may be incomplete and describing all potential malicious events in advance is impractical, hypotheses need to be put forward in order to fill in this gap.

Despite the usefulness of formal methods and approaches, digital investigation of security incidents remains scarcely explored by these methods. Stephenson [2] took interest to the root cause analysis of digital incidents and used

Copyright © 2009 SciRes.

Colored Petri Nets as formalism for modeling occurred events. The methodology may become insufficient to deal with sophisticated attack scenarios, where there is a lack of information on the compromised system that requires some hypotheses formulation. Stallard and Levitt [3] proposed a formal analysis methodology entitled semantic integrity checking analysis. It is based on the use of an expert system with a decision tree that exploits invariants relationship between existing data redundancies within the investigated system. To be usable with highly complex systems, it is imperative to have a prior list of *good state* information; otherwise, the investigator has to complete its analysis in ad hoc manner. Gladychev [4,5] provided a Finite State Machine (FSM) approach to reconstruct potential attack scenarios discarding scenarios that disagree with the available evidences. Since investigation may occur on systems that could not be completely described due to their complexity, if unknown system transitions are ignored, the event construction may freeze or its accuracy may be severely affected. Carrier and Spafford [6] proposed a model that supports existing investigation frameworks. It uses a computation model based on a FSM and the history of a computer. A digital investigation is considered as the process that formulates and tests hypotheses about occurred events or states of digital data. Additionally, the model allows defining different categories and classes of analysis techniques. A key idea in the proposed approach is that every computer has a history, which is not fully recorded or known in the practice. A digital investigation is considered as the process that formulates and tests hypotheses about occurred events or states of digital data. Willanssen [7] took interest in enhancing the accuracy of timestamp evidences. The aim is to alleviate problems related to the use of evidences whose timestamps were modified or referred to an erroneous clock (i.e., which was subject to manipulation or maladjustment). The proposed approach consists in formulating hypotheses about clock adjustment and verifying them by testing consistency with observed evidences. Later, the testing of hypotheses consistency is enhanced by constructing a model of actions affecting timestamps in the investigation system [8]. In [9], a model checking-based approach for the analysis of log files is proposed. The aim is to search for pattern of events expressed in formal language. Using this approach logs are modeled as a tree whose edges represent extracted events in the form of algebraic terms. P. Sencar and Memon [10] proposed a methodology to recover files from unallocated space of disk without the assistance of meta-data or file system table. The proposed technique assumes that files may be initially fragmented and several contiguous blocks may be scattered around the storage area. To enhance the effectiveness of file recovery, the technique looks for detecting the point of fragmentation of a file, using a sequential hypothesis testing (SHT) procedure. Peisert [11] proposed to determine what data are necessary to perform investigation and basis its idea on the use of the requires/provides model, which is commonly used for intrusion detection.

We develop in this paper, a system for Digital Forensic in Networking called DigForNet. It integrates the analysis performed by the Incident Response Team on a compromised system, through the use of a new Cognitive Map [12,13] called the Incident Response Probabilistic Cognitive Map (IRPCM), which extended the Cognitive Map proposed in [14]. DigForNet uses formal approach to identify potential attack scenarios using a formal specification language entitled, I-TLA. The formal approach allows specifying different forms of evidences. It identifies an attack scenario as a series of elementary actions, retrieved from a used library, which, if executed on the investigated system, would produce the set of available evidences. We developed in DigForNet the concept of executable specification of attack scenarios, which shows with details how an attack is performed progressively on the system and how the latter behaved during the attack. DigForNet uses I-TLC, an automated verification tool for I-TLA specifications. To alleviate any missing evidences or details related to attack scenarios, DigForNet integrates a technique for generating hypothetical actions to be appended to the scenario under construction.

Our contribution is three-fold. First, DigForNet reconciles in the same framework conclusions derived by the incident response team and theoretical and empirical knowledge of digital investigators. To the best of our knowledge, it is the first investigation system which supports such feature. Second, we proposed a new IRPCM which integrates the temporal aspect. In fact, during IRPCM construction, the appending of anti-investigation relations between concepts could make other concepts inactive. Several IRPCM snapshots could thus be obtained depending of time. Third, using the concept of hypothetical actions, DigForNet stands out from the other existing approaches and allows generating sophisticated and unknown attack scenarios. The new generated hypothetical actions could be used to extend the content of the library of attacks. Fourth, the formal techniques used by DigForNet allow supporting a collaborative working between the IRT members, and generating a formal specification useful for conducting an investigation, where a model checking-like technique could be used to automate the generation of executable specification of attack scenarios.

This paper is organized as follows. Section 2 defines the important concepts related to the digital investigation of security incidents and describes the DigForNet's methodology for reasoning about security incidents. The use of the IRPCM technique to represent the intrusion response team members' view about the security incidents is shown in Section 3. Section 4 describes I-TLA as logic for specifying evidences and identifying potential attack scenarios that satisfy them. It also shows how to pass from IRPCM to I-TLA specification. Section 5 intro-

Copyright © 2009 SciRes.

duces I-TLC as an automated verification tool for I-TLA specifications, which allows generating executable specification of attack scenarios. Section 6 illustrates an example of the use of DigForNet in investigating a real security incident. Finally, Section 7 concludes this paper.

2. Methodology of Structured Investigation

We start this section by introducing the need for digital investigation and then we describe the DigForNet's methodology.

2.1. Need for Digital Investigation

Focusing merely on restoring the system, which is the simplest and easiest method, is disadvantageous. In fact, valuable information and traces that allow understanding the attack could be removed if the compromised system is straightforwardly formatted or reinstalled. The above mentioned weaknesses in the response point up the need for conducting a post-incident digital investigation [15]. The latter can be considered as the process that allows to: 1) determine how the computer attack was performed and what are the security weaknesses and design mistakes that let the incident succeeds; 2) trace the attackers to their source to identify their identities; 3) build a proof from the collected information to bring a prosecution against attackers who committed the attack; 4) argument and underline the results with well-tested and proved methods and techniques; 5) Study the attackers' trends and motives, and take the accurate security measures to prevent future similar attack scenarios.

Since digital investigation [16,17] focuses on the investigation of an incident after it has happened, a digital evidence should be gathered from the system to support or deny some reasoning an investigator may have about the incident. Digital evidence is defined as any data stored or transmitted that support or refute a theory of how an offence occurred or that address critical elements of the alibi [16].

2.2. DigForNet Methodology

DigForNet integrates the incident response team contributions under the form of Incident Response Probabilistic Cognitive Maps (IRPCMs). An IRPCM is built with a collaborative fashion by the IRT members based on the information collected on the system. IRPCMs provide a foundation to mainly investigate and explain occurred security attacks.

DigForNet provides a formal way for reconstructing potential attack scenarios. It defines a novel logic entitled Investigation-based Temporal Logic of Actions (I-TLA), and its logic-based language entitled I-TLA$^+$. DigForNet methodology is composed of six steps organized in a waterfall model as shown in Figure 1. They are described as follows.

The first step collects evidences available within three different sources, namely the operating systems, networks, and storage systems. The second builds the IRPCM, which is nothing but a directed graph representing security events, actions and their results along with the relations between these concepts. The third step consists in extracting the sets of evidences and actions from the cognitive map for the formal specification of the potential attack scenarios. The fourth step generates a formal specification. A formal approach is necessary for this purpose. DigForNet uses logic, referred to as I-TLA, to generate a specification containing a formal description of the set of extracted evidences and actions, the set of elementary attack scenario fragments retrieved from the library of attacks, and the initial system state. During this step, DigForNet uses I-TLA to prove the existence of potential attack scenarios that satisfy the available evidences. To be able to generate a variety of attack scenarios, DigForNet considers the use of a library of elementary actions supporting two types of actions: legitimate and malicious. Malicious actions are specified by security experts after having assessed the system or appended by investigators upon the discovery of new types of attacks.

The fifth step generates executable specification [18,19] of potential attack scenarios using a model checker tool associated with the formal specification. DigForNet builds Investigation-based Temporal Logic model Checker called I-TLC composed of two sub-steps. The first works to rebuild the attack scenarios in forward and backward chaining processing, showing details of all intermediate system states through where the system progresses during the attack. The second sub-step provides a tolerance to the incompleteness of details regarding the investigated incident and the investigator knowledge. It interacts with a library of hypothetical atomic actions to generate hypothetical actions, append them to the scenarios under construction, and efficiently manage them during the whole process of generation. The library of hypothetical atomic actions is composed of a set of entries showing interaction between a set of virtual system components and a set of rules used to efficiently create hypothetical actions as a series of hypothetical atomic actions.

The sixth step uses the generated executable potential attack scenarios to identify the risk scenario(s) that may have compromised the system, the entities that have originated these attacks, the different steps they took to conduct the attacks, and the investigation proof that confirms the conclusion. These results are discussed with the IRT members in order to check the hypotheses added by I-TLC and update the initial IRPCM by: 1) omitting some concepts because they do not present an interest for the attack scenario construction, and/or 2) adding other concepts, corresponding to the hypothetical actions, to the IRPCM and linking them to the other concepts. Links in

Copyright © 2009 SciRes.

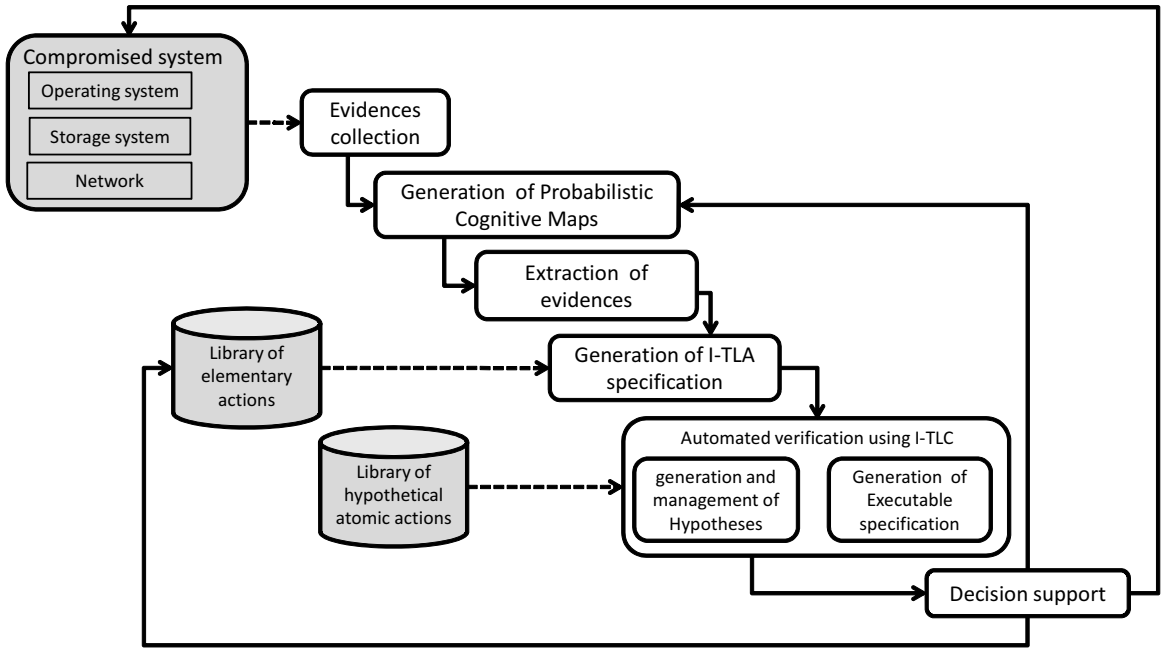

Figure 1. DigForNet methodology.

the IRPCM are deleted in the case where the concepts originating from or going to them are omitted. New links are also added to the IRPCM to link the newly added concepts. Hypothetical actions are also added to the attack library. In addition, tools collecting the evidences are enhanced to be able to detect the newly discovered vulnerabilities.

3. Intrusion Response Probabilistic Causal Maps

We have defined in [20] a category of cognitive maps to support the intrusion response activity. In this paper, we provide an extension to these cognitive maps referred to as Incident Response Probabilistic Cognitive Maps (IRPCMs) by introducing the notions of probability and activation degree of concepts, and integrating links with complex semantics. IRPCMs provide the foundation to investigate and explain security attacks which have occurred in the past and to predict future security attacks against the system. These aspects are important for negotiation or mediation between IRT members solving thus disparities which are generated by the difference in their view points and which can lead to conflicting decisions.

3.1. IRPCM Definition

An Incident Response Probabilistic Cognitive Map (IRPCM) is a directed graph that represents intrusion response team members' experience-based view about security events related to an incident. The nodes of the graph represent concepts belonging to the network security field and a set of edges representing relationships between the concepts.

IRPCM concepts can be symptoms, actions, and unauthorized results related to network security field. Symptoms are signs that may indicate the occurrence of an action. System crashes or the existence of new user accounts or files are examples of symptoms. An action is a step taken by a user or a process in order to achieve a result. Probes, scans and floods are samples of actions. An unauthorized result is an unauthorized consequence of an event (defined by an action directed to a target). For instance, an authorized result can be an increased access or a disclosure of information or a theft of resources. IRPCM concepts are labeled by values in the interval [0,1] informing about the activation of the correspondent concepts. They are also labeled by a value indicating their occurrence time.

IRPCM edges link concepts to each others. An edge e_{ij} linking concept c_i to concept c_j is labeled as (π_{ij} , q_{ij}) where π_{ij} is the predicate expressing the relation between the two nodes (examples include $<_t$, I/O , CE) and q_{ij} (taking values in]0,1]) the probability expressing the certitude degree that the relationship π_{ij} really occurred between the concepts c_i and c_j. Quantitative values are given by security experts. Notice that the semantic of the predicate π_{ij} depends on the nature of the concepts c_i and c_j. For the sake of simplicity, we consider seven cases of relationships in this paper. They are described here after.

1) Input/output relation: Let c_i be a symptom and c_j be a symptom or an action. An input/output relation, which

Copyright © 2009 SciRes.

is expressed using the predicate $\pi_{ij}=I/Q$, means that part of output of the concept c_i is the input of the concept c_j.

2) Temporal relation: Let c_i and c_j be two actions. A temporal relation, which is expressed using the predicate $\pi_{ij}=<t$, means that c_i is an action that precedes c_j.

3) Cause/Effect relation: Let c_i be an action and c_j be an unauthorized result. The cause/effect relation, which is expressed using the predicate $\pi_{ij}=CE$, means that the effect produced by concept c_i is visible through concept c_j.

4) Concealment relation: Let c_i and be an action and c_j be an action, a symptom or an unauthorized result. The concealment relation, expressed using the predicate $\pi_{ij}=$ *conceal*, means that concept c_i when it happens, leads to the hiding of concept c_j. This corresponds to the situations where the attackers execute some actions on the compromised system to hide information revealing their access to this system, or to hide the results on this system.

5) Destruction relation: Let c_i be an action and c_j be an action, a symptom or an unauthorized result. The destruction relation, which is expressed using the predicate $\pi_{ij}=destroy$, means that the occurrence of concept c_i wipes out the existence of concept c_j. This corresponds to the situation where the attacker executes some actions to destroy any trace that may inform about his access to the compromised system.

6) Forgery relation: Let c_i be an action and c_j be an action, a symptom or an unauthorized result. The forgery relation, which is expressed using the predicate $\pi_{ij}=forge$, means that the occurrence of concept c_i creates a new forged concept c_j with random time of occurrence. This corresponds to the situation where the attacker tries to deceive the investigation activity.

7) Replacement relation: Let c_i be a forged action, symptom, or unauthorized result and c_j be a concept belonging to the same category as c_i. The replacement relation expressed using the predicate π_{ij}=replace, means that the concept c_j is replaced by concept c_i when the latter has been forged by an action.

Notice that relations *conceal*, *destroy*, *forge* and *replace* corresponds to an anti-investigation activity.

3.2. Appearance-Period

Let c_i and c_j be two concepts belonging to the IRPCM having respectively occurrence times equal to t_i and t_j. Appearance period of c_j, say A_{cj} is determined as follows:

- If $\pi_{ij} = conceal, destroy, replace$ then $A_{c_j} = [t_j, t_i]$
- If $\pi_{ij} = I/O, <_t, C/E, forge$ then $A_{c_j} = [t_j, \infty]$

3.2.1. Snapshot Function

An IRPCM may vary as anti-forensic actions and relations are appended. Therefore, some concepts in the IRPCM may be invalid at a given time, and the analysis of the IRPCM becomes complex. To make the analysis simple, we need a snapshot of the IRPCM for different instants. To this end, we introduce the snapshot function. The main feature of this function is to show a sub-view of the IRPCM which hides the concepts in the IRPCM that are invisible at that time. To do so, the appearance period of concepts is exploited.

Formally, let *Vsisible*(c,t) be the function that returns the Boolean value *True* if the time t is within the appearance period of the concept c. The IRPCM snapshot at time instant t is created by deleting any concepts c in that IRPCM, such that *Vsisible*(c,t)=*false*, and all edges which are connected to c_i.

3.3. Building IRPCMs

The IRT members are responsible for building the IRPCM (second step in the DigForNet methodology). The basic elements needed in this activity are the events collected on the information system. These events may be IDS alerts; compromises of services offered by the network, or any sign indicating the occurrence of malicious or suspect actions against the network. IRT members analyze these signs and define the appropriate symptoms, actions and unauthorized results and assign the appropriate probabilities and relationships to the edges linking the defined concepts. The process of building an IRPCM has two properties: completeness (if an attack has occurred and a sufficient number of events are collected to identify this attack, then we can find an IRT able to build an IRPCM allowing to identify the attack) and convergence (if an IRPCM is built and is large enough to collect all the events related to a given attack, then the IRT must build in a finite time an IRPCM allowing to provide the right solution to protect against this attack).

The building of an IRPCM follows a methodology based on the iterative process described in the following steps:

1) Collect a first set of security events observed in the compromised system or detected by security tools.

2) Build an IRPCM based on the collected events.

3) Continue to collect security events.

4) Update the IRPCM based on the new recollected events. Events which do not belong to the previous IRPCM are added. Probable links related to the newly considered concepts are also added to the IRPCM.

5) Refine the IRPCM.

6) Update the probabilities of the links and the activation degree of the concepts.

7) If the stopping criterion is satisfied, stop the IRPCM building process; else, return to step 4.

In the second step of the above methodology, the generation and building of the IRPCM is the duty of the Incident Response Team (IRT). Two main tasks should be handled within this step. First, the IRT members should collaborate to append concepts based on their knowledge and skills, and negotiate between them to classify the appended concepts into necessary and unnecessary con-

Copyright © 2009 SciRes.

cepts. Concepts can be considered as unnecessary if they are duplicated, do not cope with the properties of the attack or the system under investigation, or are erroneous. The unnecessary concepts will thus be deleted from the IRPCM under construction. Second, the IRT members collaborate to locate concepts in the IRPCM that could be linked together and append edges. Obviously, such activity is subject to discussion and negotiation in order to correct or delete erroneous edges.

In the fifth step of the methodology, the refinement of the IRPCM is done through the analysis of the semantic of concepts. Two forms of modifications on the IRPCM take place during the refinement. The first consists in substituting a concept by a more accurate one, merging some concepts together, or segmenting a concept into many other ones to make relations more significant. While attack scenarios look different, they, in most cases, reuse techniques of attacks and actions. The IRT, which is always in charge of constructing IRPCMs for the investigated scenarios, could exploit IRPCMs related to previously resolved incidents to complete and update the current IRPCM under construction. To do so, it suffices to define patterns that allow detect similarities between similar fragments of IRPCMs. The IRT has just to detect patterns in the IRPCM under construction and find IRPCM fragments that could be integrated from the previously constructed ones.

Two criteria can be considered to decide about the end of the IRPCM building process. The first is when all the candidate actions in the library (those which have a relationship with the collected events) are present in the IRPCM. The second is based on the decision of the IRT members. If the latter agree that the IRPCM is large enough, then the building process is stopped. The IRT decision can be shared by all the members or it can be taken by a mediator (a member of the IRT in charge of coordinating activity helping solving conflicts and terminating the process).

3.4. Activation Degree of a Concept

IRPCM concepts values give indications about their activation. These values, referred to as activation degrees, belong to the interval [0,1]. We define the function *dac* assigning activation degrees to the concepts as follows:

$$\begin{aligned} dac: \quad & C \times S \to [0,1] \\ & c, s \mapsto dac(c,s) \end{aligned} \tag{1}$$

where C represents the set of concepts in the IRPCM and S stands for the set of snapshots. A concept is said to be dac-activated if its activation degree is equal to 1. In the following, we show how to build a *dac* function based on a given set of selected concepts in the IRPCM for a snapshot s. Let I be the set of concepts related to collected events of involvement in attack with respect to detected intrusions. $I = \{c_1 \cdots c_n\} \subseteq C$.

1) Let $dac(c_i, s) = 1, i = 1 \cdots n$.

2) Compute iteratively the remaining activation degrees as follows: Let F be the set of the concepts for which we have already computed the activation degree. F is initially equal to the set I.

3) Let G be the set of concepts that have a relation with one or more concepts belonging to F. $G = \{c \in C / \exists d \in F, (d,c) is an edge\}$ .Then, $dac(c,s) = sup_{d \in G} \{q_{dc} dac(d)\}$ where q_{dc} is the probability expressing the certitude degree that there is a relationship between the concepts d and c.

4) $F := F \cup G$ and return to step 3 if $F \neq \varnothing$.

In the case where the IRT members have detected malicious actions against the secured system, they start constructing the IRPCM corresponding to this situation. The concepts that represent the collected events are activated and will form the set I in this case. The activation degree of the remaining concepts is determined according to the previous algorithm.

The *dac* function is used in the third step of the DigForNet methodology to extract set of evidences. Nodes having a *dac* degree greater than a predefined threshold are extracted as evidences for the formal specification.

Notice that the activation degree of concepts may vary from one snapshot to another if some concepts are deleted or, in the contrary, added to the current snapshot. In the first case, a concept c_m is deleted from a given snapshot of the IRPCM. If c_f is a concept to which c_m is directly linked, then the activation degree of the concept c_f is reduced if the activation degree of c_m is the most important over the set of concepts directly linked to c_f. In the second case, if c_m is a concept which is added to the current snapshot, we distinguish three sub-cases: 1) the new concept is not evidence and has no concepts directly linked to it. In this case the *dac* value of the new concept is unknown and must be set by the investigation team; 2) the new concept is evidence. In this case its *dac* value is set to one; 3) the new concept is not evidence and there are concepts directly linked to it. In this case, the *dac* value of the new concept is calculated according to the previous algorithm. Having determined the *dac* value of the new concept c_m and if we represent by P the set of concepts to which c_m is directly linked, then the for every concept c in P, the activation degree increases if c_m has the highest activation degree over those directly linked to c.

During IRPCM construction, it may happen that in some snapshots, some concepts constitute evidences, while they did not in the preceding IRPCM snapshot. In this case, we set to one the activation degree of these concepts and, using the previous algorithm, we update the *dac* values of the concepts to which these evidences are directly or indirectly linked. Conversely, if some anti-forensic relations appear in the new snapshot show-

Copyright © 2009 SciRes.

ing that some data sources were affected by attacks to alter the stored evidences, the dac values of concepts related to evidences collected from these sources should be reduced. Consequently, the algorithm is re-executed to update the dac values in the new IRPCM snapshot.

4. Generation of a Formal Specification of Attack Scenarios

The Investigation-based Temporal Logic of Actions, I-TLA [21], is a logic for the investigation of security incidents. It is a extension to the Temporal Logic of Actions (TLA [22]). I-TLA defines a theoretical framework for: 1) modeling and specifying evidences left by intruders further to the occurrence of a security incident; 2) supporting advanced description and specification of potential happened attack scenarios as a series of elementary attacks, extracted from the library of attacks, that if assembled together, would satisfy the available evidences. Similarly to TLA, I-TLA allows to reason about systems and their properties in a uniform logic formalism. I-TLA is provided with I-TLA$^+$, a highly expressive formal language that defines a precise syntax and module system for writing I-TLA specifications. I-TLA will be used in this paper to generate a specification describing potential attack scenarios that satisfy the available set of evidences.

In the sequel, we focus on describing the different forms of evidences supported by I-TLA, showing how they can be specified and how they should be satisfied by the expected attack scenario. The reader is referred to [21] for more details of I-TLA and I-TLA$^+$ and a complete semantic and syntactic description.

4.1. Modeling Scenarios and Evidences in I-TLA

I-TLA is typeless and state-based logic. It allows the description of states and state transitions. A state, while it does not explicitly appear in a I-TLA specification formula, is a mapping from the set of all variables names to the collection of all possible values. An I-TLA specification ϕ generates a potential attack scenario in the form of: $\omega = \langle s_0, s_1, ..., s_n \rangle$, as a series of system states s_i ($i = 0$ to n) that satisfies all available evidences. I-TLA supports four different forms of evidences, namely history-based, non-timed event-based, timed event-based, and predicate-based evidences. A state-based representation of attack scenarios allows a security expert to observe how its system progresses during the attack and how it interacts with the actions executed in the scenario.

4.1.1. History-Based Evidences

Typically, security solutions do not have direct access to all system components. Some of them are able to provide evidences as histories of the values of the monitored system variables, during the spread of an attack scenario. These security solutions cannot realize that the system has progressed or not from a state to another if the value of the monitored component is either blind, or does not change. I-TLA encodes a history-based evidence, say E, as an observation over a potential attack scenario ω, generated by $Obs(\omega)$ ($Obs()$ is the observation function that characterizes the ability of a security solution to monitor the history of the system during an attack scenario). It uses a labeling function that allows a third party to only see limited information about states of an execution. Since a state is a valuation of all system variables, a labeling function allows to either:

- Totally observe the content of variable value. Variable v is visible and its value is interpretable by the observer. It represents a system component whose modification is monitored by some security solution.
- Observe a fictive value instead of the real variable value. Variable v is visible but not interpretable by the observer, meaning that its variation does not bring any supplementary information to an observer. It can represent an encrypted data whose decryption key is unknown by the observer.
- Observe empty value. Variable v is completely invisible, such that none information regarding its value could be determined. It represents a system component which is not monitored by any security solution.

$Obs(\omega)$ is obtained by following two steps:

1) Transform each state s_i to $\hat{s}_i$, by hiding some of the details it provides. $\hat{s}_i$ is obtained from s_i by making the value of every system variable v in s_i to be:

a) Unmodified. In this case the variable is visible and its value is interpretable by the observer. It represents a system component whose modification is monitored by some security solution;

b) Equal to a fictive value fictive value. In this case the variable is visible but not interpretable by the observer, meaning that its variation does not bring any supplementary information to an observer. It can represent an encrypted data whose decryption key is unknown by the observer;

c) Equal to an empty value, denoted by ε. In this case, the variable is completely invisible, such that none information regarding its value could be determined. It represents a system component which is not monitored any security solution.

2) Delete any $\hat{s}_i$ which is equal to null value (i.e., all values are invisible) and then collapse together each maximal sub-sequence $\langle \hat{s}_i, ..., \hat{s}_j \rangle$ such that $\hat{s}_0 = ... = \hat{s}_i$, into a single $\hat{s}_i$.

Taking into consideration the availability of a history-based evidence E, consists in generating, an attack scenario ω such that $Obs(\omega)=E$.

Copyright © 2009 SciRes.

4.1.2. Ordering of Observations

A step in the scenario may not change all the values of the system variables. As the scope of the observations differs, they may not allow noticing that the system has progressed during the attack at the same time. I-TLA allows to specify for two given observations, which one will vary first (respectively last) when the attack scenario starts (respectively finishes). Consider the following example involving an attack scenario ω and two observations $OBS = [e_1,...,e_n]$ and $OBS' = [e_1^{'},...,e_m^{'}]$, generated by observation functions $Obs()$ and $Obs'()$, respectively. OBS is said to be an observation that allows to notice the occurrence of an incident before observation OBS', if and only if: $\exists \omega_x$ such that: $\omega = \omega_x \omega_y \wedge obs_1(\omega_x) = e_1 \wedge obs_2(\omega_x) = [e_1^{'},...,e_j^{'}]$ for some j $(1 < j \leq m)$.

4.1.3. Non-Timed Events-Based Evidences

As the length of observations is different from the length of an attack scenario, reconstructed attack scenarios may differ by the manner in which observations are stretched and aggregated together to generate intermediate states of the execution. I-TLA defines non-timed events based evidences in the form of predicates over I-TLA executions, which specify the modification pattern of variables values through an execution. For instance, the execution predicate *AtSameTime*, states that state predicate p_1 switches to true at the same time the state predicate p_2 switches to false.

$$AtSameTime(p_1,p_2) \triangleq \forall \langle s_i, s_{i+1} \rangle \in \omega : (s_i \nvDash p_1 \wedge s_{i+1} \vDash p_1) \Rightarrow s_i \vDash p_2 \wedge s_{i+1} \nvDash p_2) \quad (2)$$

Taking into consideration the availability of a non-timed event-based evidence E, is amount to generate an attack scenario ω such that $\omega \vDash E$.

4.1.4. Timed Events-Based Evidences

Starting from a set of available alerts, an investigator can extract some indications related to occurred events. I-TLA defines timed event-based evidence $E = [A_0,...,A_m]$ as a set of ordered actions (A_0 to A_m) that should be part of an expected execution. While the order in which events appear should be respected, there is no need that these events be contiguous. Given a timed event-based evidence $E = [A_0,...,A_m]$, an execution $\omega = \langle s_0,...s_n \rangle$ satisfies evidence E if and only if: $\forall (A_x, A_{x+1}) \in E : \exists (s_i, s_{i+1}) \in \omega$ such that $A_x(s_i, s_{i+1}) =$ *true* and $A_{x+1}(s_j, s_{j+1}) = true$ for some $j \geq i+1$.

4.1.5. Predicate-Based Evidences

With regards to the security response team's members, an unexpected system property is a preliminary argument supporting the incident occurrence (e.g., the integrity of a file was violated). I-TLA defines predicate-based evidence as a predicate, say E, over system states, that characterizes the system compromise. An execution ω satisfies evidence E if E divides ω into two successive execution fragments ω_1 and ω_2 (ω can thus be written as $\omega=\omega_1\omega_2$). ω_1 is composed of secure states ($\forall s \in \omega_1 : s \nvDash pr$), while ω_2 is composed of insecure system states ($\forall s \in \omega_2 : s \nvDash pr$).

4.1.6. Illustrative example

The following example clarifies the use of I-TLA in digital investigation, and illustrates the mechanism of handling evidences during the construction of potential attack scenarios. We consider a system under investigation which is specified by three variables *x*, *y*, and *z*. The initial system state, described in advance, is the state defined by variables *x*, *y*, and *z* are all equal to *0*. The library of elementary actions contains two actions A_1 and A_2 that can be executed by the system.

$$A_1 \triangleq \begin{aligned} x' &= x \\ y' &= y+1 \\ z' &= z+2 \end{aligned} \quad (3)$$

$$A_2 \triangleq \begin{aligned} x' &= x+1 \\ y' &= y \\ z' &= z/2 \end{aligned} \quad (4)$$

Action A_1, for instance, keeps the value of variable *x* in the new state unchanged with respect to the previous state, and sets the values of *y*, and *z* in the new state 1 and 2 higher than its values in the old state, respectively.

Three different evidences are provided. The two first ones represent history-based evidences, defined as $E_1 = \langle 0\varepsilon\varepsilon, 1\varepsilon\varepsilon, 2\varepsilon\varepsilon \rangle$ and $E_2 = \langle \varepsilon 0\varepsilon, \varepsilon 1\varepsilon, \varepsilon 2\varepsilon, \varepsilon 3\varepsilon \rangle$, where ε stands for the invisible value. These evidences are generated by observation functions $obs_1()$ and $obs_2()$, respectively. The first observation function $obs_1()$, allows a security solution to only monitor variable *x*, meaning that, when it is applied to a state s, makes the value of y and *z* both equal to ε, and keeps the values of variable *x* unchanged. The second observation function $obs_2()$ allows a security solution to only monitor variable *y*. The ordering of observations indicates that observation provided by $obs_2()$ allows to notice the occurrence of an incident before the observation provided by $obs_1()$. The third evidence E_3, is provided as a predicate-based evidence defined as $E_3 \triangleq z \geq 1$. The fourth evidence E_4, defined as $E_4 \triangleq \forall \langle s_i, s_{i+1} \rangle \in \omega : (s_i \nvDash p_1 \wedge s_{i+1} \vDash p_1) \Rightarrow s_i \vDash p_2)$, is an non-timed evidence, stating that predicate $p_1 \triangleq x = 1$, false in a state s_i, could not switch to true in the next state

Copyright © 2009 SciRes.

s_{i+1}, unless predicate $p_2 \triangleq z \neq 4$ is true in that state. Finally, evidence E_5, indicates that sequences of events (A_1, A_2) is part of the attack scenario.

Figure 2 shows how I-TLA guarantees the satisfaction of evidences during the construction of the potential attacks. Two potential attack scenarios satisfying the available evidences are provided by I-LA, namely ω_1 and ω_2. The first scenario ω_1 is described as ω_1=<s_1, s_3, s_4, s_7, s_{12}, s_{15}>, and consists in consecutively executing the five following actions $A_1 \rightarrow A_1 \rightarrow A_1 \rightarrow A_2 \rightarrow A_2$. The second scenario ω_2 is described as ω_1=<s_1, s_3, s_5, s_9, s_{11}, s_{18}>, and consists in a consecutive execution of the five actions $A_1 \rightarrow A_2 \rightarrow A_1 \rightarrow A_1 \rightarrow A_2$.

Starting from state s_1, I-TLA cannot execute action A_2 as it moves the system to a state that does not satisfy the ordering of observations. In fact, the sub-scenario <s_0, s_1> is observed by obs_1() as <0εε,1εε> and by obs_2() as <ε0ε>. Thus, the event A_2 is detected by E_1 but not by E_2. Starting from state s_4, I-TLA does not execute action A_2 as it moves the system to a state that violates evidence E_4. State s_8 could not be considered in the construction process as it violates predicate E_3. In fact, the predicate $p1$ has become already true in state s_3 and should not change to false in state s_8 again. I-TLA discards states s_{13}, s_{14}, and s_{19} as each one of them would create an execution that violates evidence E_2 if appended to the scenario under construction. In the same context, state s_{16} is also not added to the scenario under construction as it creates an execution that violates E_1.

4.2. Handling Anti-Investigation Attacks

To elude the process of digital investigation, a seasoned attacker may try to conduct an anti-investigation attack [23, 24] to remove, hide, obfuscate, or alter available evidences after breaking into the system. Available techniques include deletion of relevant log entries, installation of root-kits, steganography, and even wiping of disks [25] to disable any further recovery.

Let *obs*(-) be an available observation function, and *OBS* be a history-based observation which corresponds to the output of *obs*(-) when executed on the attack scenario under progression, say ω. Formally, an anti- investigation attack represents any action which moves the system to some state, say s_j in ω, and does not only append $obs(s_j)$ to *OBS*, but also affects *OBS* to modify any content related to $obs(s_0,\ldots,s_{j-1})$.

We remind that I-TLA reconstructs attack scenarios by executing an action unless it satisfies all the available history-based evidences. Let s_i be the current state reachable from the initial state s_0 through the execution <$s_0,\ldots,s_{i-1}$>. If an I-TLA action A is executed from state s_i to produce state s_{i+1}, the execution obtained after reaching the new state should satisfy the available observation. Formally, $obs(\langle s_0,...,s_{i+1}\rangle) \subseteq OBS$. We demonstrate in the following the impact of the anti-investigation attack on the process of attack scenarios reconstruction in I-TLA using the regular definition of observation functions.

4.2.1. Example

We consider a system modeled using two variables p and l which are related to the user granted privilege and the content of the system log file, respectively. Variable p can take three values: 0, 1, and 2 which stand for no access, unprivileged access, and privileged access, respectively. As the log file is typically accessed in append mode, variable v takes a series of values representing the commands executed on the system. These values are included in chronological order of their execution. $\omega = \langle s_0,..,s_5\rangle$ represents an attack scenario composed of five states, where actions A_1, A_2 and A_3 stand for the execution of arbitrary commands. Every one of these actions appends an entry to the tail of the log file. A_4 consists in exploiting vulnerability on the system to get a privileged access. Action A_5 is an anti-investigation attack. It consists in getting a privileged access on the system and altering the content of the log file by deleting the entry corresponding to the execution of action A_2. The attack scenario ω is described as a series of six states, where every state is a valuation of the two variables, and edges are labeled by the executed action.

$$[1,\langle -\rangle] \xrightarrow{A_1} [1,\langle -, Act_1\rangle] \xrightarrow{A_2} [1,\langle -, Act_1, Act_2\rangle] \xrightarrow{A_3} [1,\langle -, Act_1, Act_2, Act_3\rangle] \xrightarrow{A4}$$

$$[2,\langle -, Act_1, Act_2, Act_3\rangle] \xrightarrow{A_5} [2,\langle -, Act_1, Act_3\rangle]$$

We consider a security solution which is modeled by the observation function *obs*(). The latter allows observing the current executed commands on the system by looking for new entries appended to the log file. Formally, $obs(s)=tail(s(log))$, where $tail(x)$ returns the last entry in x. Using the regular definition of observation function *obs*(), the history-based evidence generated by the security solution further to the execution of the attack scenario is given by:

$$\begin{aligned} &obs(\omega) = \\ &\langle obs(s_1), obs(s_2), obs(s_3), obs(s_4), obs(s_5), obs(s_6)\rangle \\ &= \langle -, Act_1, Act_2, Act_3\rangle \end{aligned} \quad (5)$$

Starting from this definition, it is expected that the provided observation content will be in the form of <-, Act_1, Act_2, Act_3,>. However, since this history-based evidence is provided by the content of the log file, which is retrieved after the attack, only the content <-, Act_1, Act_3,> will be visible. The difference between the expected and the available output is due to the execution of the anti-investigation attack.

If this history-based evidence is considered during the reconstruction of the attack scenario, starting from state s_2,

Copyright © 2009 SciRes.

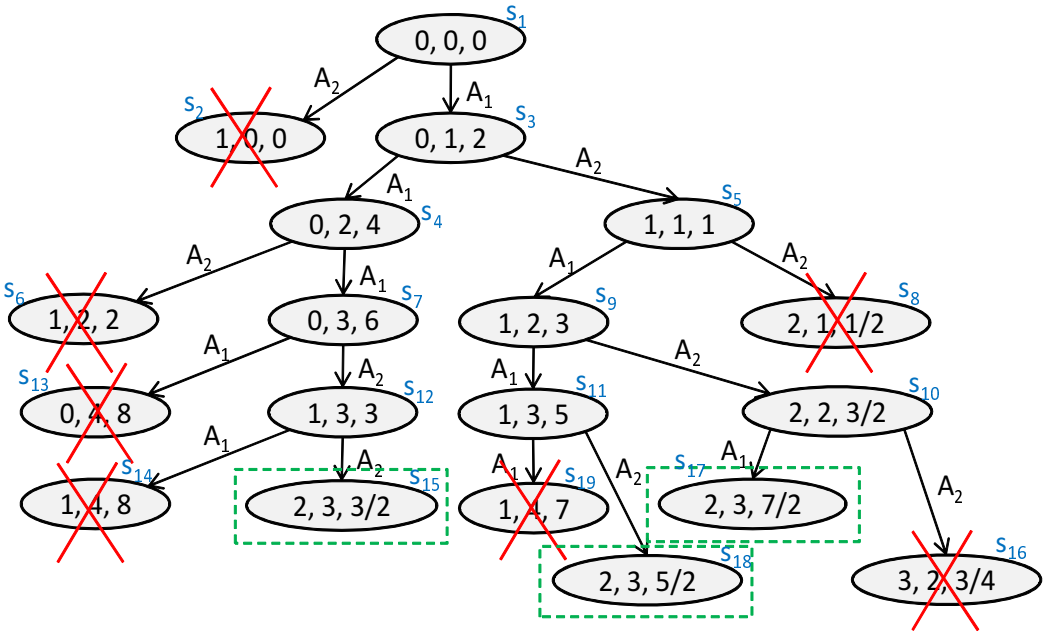

Figure 2. I-TLA attack scenario specification: an illustrative example.

action A_2 cannot be executed since it does not provide a state whose observation is included in the evidence. In other words, there is no entry in the log file after the one corresponding to the execution of A_1, which shows an indication regarding the executed action A_2.

Starting from this statement, we describe in the sequel a new observation function which allows coping with anti-investigation attacks.

4.3. Observations

Let $\{v_1,...,v_n\}$ represents the set of system variables. A variable in this set could represent a system component which is accessed in append mode (e.g., log or alert file, raw traffic capture) and takes a value or a series of values. Let $v^x(s)$ and $Card(v(s))$ represents the x^{th} value, and the number of values, in the series $v(s)$, respectively. We denote by $v(s_i) \circledast v(s_{i-1})$ the operation which consists in superimposing the series $v(s_i)$ on the series $v(s_{i-1})$ while keeping elements in $v(s_i)$ and discarding those situated beyond the limit of the intersection. Formally, $v(s_i) \circledast v(s_{i-1}) = [v^1(s_i),...,v^y(s_i)]$ where $y = min[Card(v(s_i)), Card(v(s_{i-1}))]$.

We denote by $obs^*(\omega)$ a new observation function over the executed attack scenario which allows capturing the situation where an anti-investigation attack has been conducted. Formally,

$$obs^*(\omega) = \langle obs(s_n \circledast ... \circledast s_0), obs(s_n \circledast ... \circledast s_1),..., obs(s_n)) \quad (6)$$

where

$$s_i \circledast s_{i-1} = [v_1(s_i) \circledast v_1(s_{i-1}),..., v_n(s_i) \circledast v_n(s_{i-1})] \quad (7)$$

By applying $obs^*(-)$ function on the scenario ω provided in the example of subsection 4.2, we obtain $obs^*(\omega)$= <-, A_1, A_3>. The output of this function is equal to the content retrieved from the system after the execution of the attack scenario which included an anti-investigation attack.

Theorem 1: Given an executed attack scenario ω, and an observation function $obs(-)$. If $obs^*(\omega) \neq obs(\omega)$, the attack scenario ω includes an anti-investigation attack.

Proof: We suppose that $obs^*(\omega) \neq obs(\omega)$ and there is no anti-investigation attack in the scenario ω. In the following we will disapprove this proposition.

Let v be an observable variable (with regard to $obs(-)$ function). Typically, since the variable v is in append mode, and the modifications are introduced to the tail of the series, the x^{th} value in $v(s_i)$ should be the one corresponding to the x^{th} value in $v(s_{i-1})$. In the absence of anti-investigation attack, none action executed from state s_i would modify the x^{th} value in $v(s_{i-1})$. Formally, the following condition should be satisfied:

$$\forall x \in [1..Card(s_i)] : v^x(s_{i-1}) = v^x(s_i) \quad (8)$$

Therefore, $v(s_i) \circledast v(s_{i-1}) = v(s_{i-1})$. Assuming that the investigated system is modeled using only variable v and the attack scenario is composed of two states s_i and s_{i-1}, we obtain $obs(s_i \circledast s_{i-1}) = obs(s_i)$ and $obs^*(\omega) = obs(\omega)$. The proposition is therefore disapproved.

4.4. From IRPCM to I-TLA Specification

Starting from the IRPCMs built by the IRT, useful in-

Copyright © 2009 SciRes.

formation, in the form of symptoms, unauthorized results, or actions, will be extracted and used to formally describe different type of evidences with I-TLA. We denote by useful information, any concept in the IRPCM having a degree of activation value that exceeds some predefined threshold, denoted by extraction threshold.

Symptoms are typically extracted from log files, traffic capture, or even keystrokes. They can be traduced to history-based evidences by transforming the whole content of the log file (including the record indicating the symptom itself) into an I-TLA history-based evidence. Symptoms extracted from alerts files indicate the occurrence of an events whose position in the reconstructed attack scenario cannot be determined. They will typically be transformed to non-timed I-TLA based evidence.

Actions selected from an IRPCM represent steps taken by a user or a process in order to achieve some result. A well intentioned reader has noticed that actions in the I-TLA library and actions in the IRPCM may not have the same form, and are not of the same granularity. In fact, an IRPCM action can be traduced to one or several consecutive I-TLA actions. In this context, for every selected IRPCM action an investigator has to extract sequence of elementary actions from the I-TLA library. The different obtained sequences will represent timed events-based evidences.

Unauthorized results represent unauthorized consequence of events. They are traduced to I-TLA predicate-based evidences. An investigator has to identify the system variable affected by the unauthorized consequence and then use it to describe the evidence.

Since the attack scenario may integrate anti-investigation attacks, the investigator has to locate in the IRPCM the set of concepts that are linked by anti-investigation relations (i.e., conceal, destroy, forge, and replace). Since anti-investigation attacks are executed to compromise evidences, the investigator has to determine which of the system variables, specified with I-TLA, describe the content of these compromised evidences. After that, it has to select observation functions that are defined to observe the content of these affected variables. This feature is highly essential for the reconstruction of the attack scenarios.

5. Executable Scenarios Generation Using I-TLC

To automate the proof in the context of digital investigation and generate executable attack scenarios showing with details how the attack was conducted and how the system progressed for each action part of the scenario, I-TLC [21], a model checker for I-TLA$^+$ specifications can be used. I-TLC is somehow an extension to TLC, the model checker of TLA$^+$ specification. It works by generating an optimized directed graph of states representing the space of possible scenarios generated from the I-TLA$^+$ specification. Despite checking that some types of computation are impossible as they violate safety properties, I-TLC aims to reconstruct execution (i.e., potential attack scenarios) that satisfy each form of evidences supported by I-TLA. I-TLC provides a novel concept entitled hypothetical action, defines techniques for its generation and management, and improves the representation of states. The directed graph is built by ensuring that a given node is reachable under optimal sets of hypothetical actions.

5.1. I-TLC's States Representation

I-TLC represents a node in the graph as a tuple of two information: *node core* and *node label*. The core of a node represents a valuation of the entire system variables, and the node label represents the potential sets of hypothetical actions under which the node core is reached. A reading of the node label indicates a) the state of the system in the current node, and b) the alternatives (hypothetical action sets) under which the system state is reachable.

$[(1,3),\{\{H_1,H_4\},\{H_2,H_3\}\}]$ represents an example of a node which can be represented by the graph generated by I-TLC. (1,3) is the node core, $\{H_1,H_4\}$ and $\{H_2,H_3\}$ represent the set of hypothetical actions under which the node core is reachable, and $\{H_1, H_4\}$, $\{H_2, H_3\}$ is the node label. This representation means that the system state (1,3), representing a valuation of the two variables x and y, respectively, is reachable under one of the two sets of hypothetical actions $\{H_1,H_4\}$ or $\{H_2,H_3\}$.

5.2. Generation of Hypothetical Actions

Generation of potential attack scenarios may fail if the library of actions is incomplete. In fact, for the particular case of attack scenarios that involve the use of unknown techniques, the system may come at some state while being unable to reach another state that if appended to the scenario under construction, will make it satisfy all the available evidences. To alleviate this issue, I-TLC tries to generate a hypothetical action and append it to the graph under construction, whenever available evidences are not completely satisfied.

The idea behind the generation of hypothetical actions is based on the fact that unknown actions can be generated if additional details about internal system components (i.e., those abstracted by the specification) are available. This detail involves a description of how these internal system components are expected to behave (if atomic actions are executed on them) and how they depend on each other. These internal system components are modeled by a specific set of variable denoted by internal variables. The other variables specified by I-TLA are denoted by external variables.

Semantically, a hypothetical action is true or false for

Copyright © 2009 SciRes.

a pair of states <*s*, *t*>. Syntactically, a hypothetical action is modeled as a series of hypothetical atomic actions, executed one after the other from state *s* to move the system to state *t*. It is defined in the following form $H = m_{ie}h_0 \rightarrow ... \rightarrow h_n m_{ei}$. m_{ie} defines a mapping from the external variables values to the internal variables values in state *s* and m_{ei} defines a mapping from the internal variables to the external variables in state *t*. The set of h_i (*i* from 0 to *n*) represents executed hypothetical atomic actions. A hypothetical atomic action h_i only modifies a single internal variable, and represents a relation between two consecutive internal system states. During hypothetical actions generation, I-TLC needs access to the library of hypothetical atomic actions. This library describes all the potential hypothetical atomic actions that can be executed on the investigated system.

During scenarios generation, several hypothetical actions may be appended whenever needed. I-TLC manages hypotheses following the two key ideas. First as hypotheses are not completely independent from each others and some hypotheses are contradictory, I-TLC avoids reaching a state under a contradictory situation. In this context, the library of hypotheses indicates potential contradictory sequence of hypothetical atomic actions. Second, in order to ensure that generated hypothetical actions are at the maximum close to real actions performed on the system, I-TLC defines techniques to refine the selection of hypothetical atomic actions.

5.3. Generation of Anti-Investigation Attacks

Typically, when an I-TLA action is to be executed, I-TLC verifies whether it satisfies all available evidences, especially history-based observations. Similarly to the case of unknown actions (as discussed in the previous subsection), the generation of potential attack scenarios may fail if the history-based observations were compromised using an anti-investigation attack. To cope with such an issue, I-TLC handles separately actions which modify the compromised evidences (with regard to history-based observations detected by I-TLA in the previous phase to be compromised using anti-investigation attacks).

Let $\mathcal{O}$ be the set of observations compromised by anti-investigation attacks, and *V* be the set of variables affected by these attacks. I-TLC could execute, during the reconstruction of the attack scenario, an action, say *A*, which do not create states satisfied by the available observations in $\mathcal{O}$, provided that the executed action modifies at least a variable in *V*. In the sequel, an action which satisfies the above conditions will be entitled *Prep-anti-investigation* action.

To support the generation of *Prep-anti-investigation* actions, heuristics can be used so that only accurate actions will be integrated to the attack scenario under construction. These heuristics exploit the values that could be taken by some other variables in the execution. For instance:

- Execute an action *A* if one of the collected evidences, namely the history-based evidences, shows that later the user will get a privileged system access. Such condition would mean that an anti-investigation attack can potentially be executed.
- Discard the attack scenario under construction if the number of states between the first generation of action *A* and the execution of the anti-investigation action has exceeded a threshold.

While staring from *the first generation of Prep-anti -investigation* action, the generated attack scenario will no longer satisfy the available observations, I-TLC should verify later that, further to the execution of some I-TLA action, which will typically be an anti-investigation attack (included in I-TLA as evidence), the attack scenario under construction, say ω, becomes satisfied by all evidences. For a completely generated attack scenario, say ω, which included, at some step in the execution, an anti-investigation action, I-TLC should verify that $obs(\omega) \neq obs^*(\omega)$.

5.4. Inferring Scenarios with I-TLC

To generate potential scenarios of attacks, DigForNet uses I-TLC Model Checker, which follows three phases. The reader is referred to [21] for a detailed description of I-TLC algorithms.

5.4.1. Initialization Phase

During this step, the generated scenarios graph is initialized to empty, and each state satisfying the initial system predicate is computed and then checked whether it satisfies the system invariants and the set of evidences. In that case, it will be appended to the graph with a pointer to the *null* state, and a label equal to $\varnothing$ (as no hypothetical action is generated).

5.4.2. Forward Chaining Phase

The algorithm starts from the set of initial system states, and computes in forward chaining manner the entire successor states that form scenarios satisfying evidences described in I-TLA. Successor states are computed by executing an I-TLA action or by generating a hypothetical action or Prep-anti-investigation action, and executing it.

When a new state is generated, I-TLC verifies if another existing node in the graph has a node core equal to that state. If the case is false, a new node, related to the generated state, is appended to the graph under construction, and linked to its predecessor state. If the case is true, the label of the existing node is updated so that it embodies the set of hypothetical actions under which the new system state is reachable. During label update, I-TLC ensures that each node label is provided with the

Copyright © 2009 SciRes.

following properties: soundness (a node holds the set of hypothetical actions under which its core is reachable), consistency (none set of hypothetical actions in the node label is an inconsistent or contradictory one), completeness (every set of hypothetical actions in the node is a superset of some other hypothetical actions), and minimal (none set of hypothetical actions is a proper subset of any other). If the scenario yielding the new generated state satisfies all the evidences, the system state is considered as a terminal state.

5.4.3. Backward Chaining Phase

All the optimal scenarios that could produce terminal states generated in forward chaining phase and satisfy the available evidences, are constructed. This helps obtaining potential and additional scenarios that could be the root causes for the set of available evidences. During this phase, the algorithm starts with a queue holding the set of terminal states generated in forward chaining phase. Afterwards, and until the queue becomes empty, the tail of the queue is retrieved and its predecessor states is computed. The new generated states are managed and appended to the graph under construction with the same manner followed in forward chaining phase.

All potential scenarios are supposed to be generated by I-TLC. The only exception may occur due to the lack of actions in the library of elementary actions. Nonetheless, the use of hypothetical actions concepts allows alleviating this problem.

6. Case Study

To demonstrate how DigForNet works, we provide in the following a case study related to the investigation of a compromised Linux Red Hat 7.2 operating system, which was deployed as a Virtual Honeypot in a VMWare session. The compromised system was suspended with VMWare immediately after the attack and a live image was created and posted by the Honeynet Project[1] for investigation. This case study deals with an investigation of a live system, the attack is highly complicated and requires advanced digital investigation skill knowledge, and the conducted scenario integrates several anti-investigation actions. In this case study, we will start by describing the attack. Then, we will show the use of DigForNet to investigate such incident.

6.1. Attack Description

First, the attacker probed the HTTP server from the machine identified by the IP address 213.154.118.219. Then, the attacker tried to exploit the Apache SSL handshake bug. Using this vulnerability, he gained a remote access as the Apache user. After that, he escalated his privilege and gained root access. At this level, the attacker conducted many attempts to install a rootkit. Only one of these attempts has succeeded. The following paragraphs describe the rootkits installation attempts.

The attacker has downloaded the tarball rk.tar.gz from geocities.com/mybabywhy/rk.tar.gz. Obviously, he then installed the rk.tar.gz. This install operation infected some binary files on the system, including ifconfig, ls, netstat, ps and top, and saved their original version in /usr/lib/libshtift. When the install script of rk.tar.gz finished the installation process, some system files (such as /bin/ps) have been replaced; mails with information about the system have been sent to mybabywhy@yahoo.com and buskyn17@yahoo.com; new unknown processes have been run as daemon; and the log file has been deleted to hide the attacker actions. After this, the hacker downloaded other tools including abc.tgz, an installation script for the current SSH server; and mass2.tgz, which is an exploit used to hack the server. However, the attacker has failed to stop the SSH daemon and has installed an SSH server under the file name "smbd -D". The attacker does not even know the backdoor password. So, he carried out a novel attempt. He downloaded adore rootkit and tried to install it. But the install operation failed.

The attacker did not give up. He again gained a root access. This time, the attacker used the program gods (a shell script from izolam.net), to download adore LKM and an SSH server. After this, the attacker has installed the SucKIT rootkit using the installation script inst. This time, the rootkit installation has succeeded. The attacker also run xopen and lsn programs and moved /lib/.x/.boot from /var/tmp/.boot. After this, the attacker has connected to the FTP server identified by the IP address 63.99.224.38. Then, the file /root/sslstop.tar.gz has been moved from /lib/.x/s.tgz. It contains the sslstop program which modifies httpd.conf to disable the SSL support. The program sslport modifies httpd.conf to change the default SSL port (443) to another port (3128 in this case). The primitive HAVE_SSL has been replaced by HAVE_SSS in /etc/httpd/conf/httpd.conf. This indicates that sslstop has been run. In addition, the attacker downloaded psyBNC from www.psychoid.lam3rz.de/ psybnc and installed it. This program is used to hold open an IRC connection and run a proxy IRC in order to hide the user's IP address. Using psyBNC, the user *sic* has connected from sanido-09.is.pcnet.ro to fairfax.va.us. undernet.org, an IRC server. He has created an account named redcode.

6.2. IRPCM Construction

The generated IRPCM is given by Figure 3. Actions, symptoms, and unauthorized results in this IRPCM are depicted by plain, dashed, and dotted ellipses, respectively.

To construct the IRPCM, we used the evidences which

[*] Honeynet Project-Scan of the Month #29 http://old.honeynet.org/scans/scan29/

Copyright © 2009 SciRes.

were collected on the compromised system. Three concepts in the form of symptoms, namely S_1, S_2, and S_3, are initially appended to the IRPCM. Symptom S_1 indicates that the httpd log file contains suspicious entries showing potential exploit of ssl. Symptom S_2 indicates the existence of suspicious connections to the web server from 213.154.118.218 in the /var/log/ messages. Symptom S_3 shows that the web server banner is indicating a vulnerable version of Apache/OpenSSL. These three symptoms are linked to the concept A_1 which represents the action "Execution of mod_ssl/OpenSSL exploit". The latter action leads to the creation of the following unauthorized result, denoted by U_1 "Unauthorized access to the system with privileged rights" meaning that the intruder can execute some commands on the compromised system. Therefore, an edge is appended to the IRPCM from the concept A_1 to the concept U_1, creating a Cause/Effect relation.

The attacker reconnected to the system from the IP address 213.154.118.218. In the IRPCM this action is defined by the concept A_2, succeeds the action A_1, and precedes the action A_3 which represents a tentative to install the Adore rootkit. Action A_3 is vindicated by the content of log files and some mails from Apache indicating a failed installation of the Adore rootkit. In this context, the symptom S_5 is linked to the action A_3.

Always using the privileged access, the attacker downloaded rk.tar.gz from geocites and installed the rootkit it contains. This is shown by the action A_4 in the IRPCM. The latter is vindicated by the content of swap-colon.txt (symptom S_8 in the IRPCM). This rootkit leads to the unauthorized result U_2 which indicates that an unauthorized installation of programs on the system was performed. The attacker succeeded to install other programs such as a port scanner called sl2. Such activity is represented by action A_5 in the IRPCM. It is vindicated by some entries in the swap-colon.txt file too. By the completion of the installation, the attacker erased the log installation history of the tools contained in rk.tar.gz. Since this behavior represents an anti-forensic attack, the concept A_4 is linked to the concept S_9 using a destruction relation. The investigation process did not show any further use of this rootkit.

After this, the attacker used his privileged access to run the /lib/.x/hide script (action A_6 in the IRPCM) in order to destroy the symptoms "gods is running" and "inst is running". Two destruction relations are therefore created from the action A_6 to the symptoms S_{10} and S_{11}, respectively. The two latter symptoms give a proof regarding the execution of actions "Install SucKIT" and "Download SucKIT", respectively. In reality, the attacker installed the SucKIT rootkit as proven by the / partition analysis which shows the use of gods, a script used to download SucKIT, and inst, a script used to install this rootkit. SucKIT installation led to the unauthori-

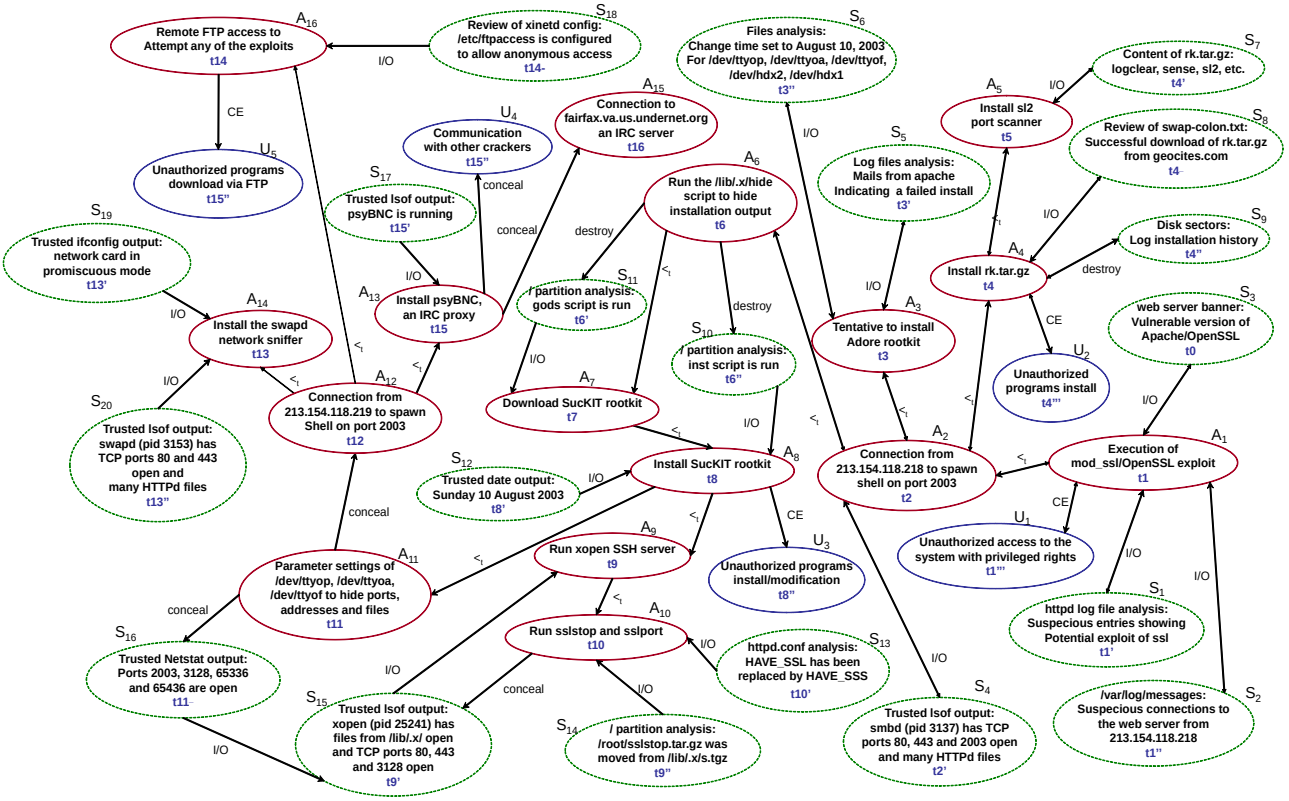

Figure 3. IRPCM related to the attack against the VMWare Linux honeypot.

Copyright © 2009 SciRes.

zed result "unauthorized programs install/ modification", namely U_3. This rootkit was installed on 10 August 2003 as indicated by a trusted version of the date command. This information was provided by symptom S_{12} which is linked to the action A_8. The action representing the SucKIT installation was followed by action A_9 denoted by "run xopen SSH server". This action is vindicated by the symptom S_{12} which is provided by the output of a trusted version of the lsof command. The latter shows that xopen is running on the compromised system. This output shows also that SSL is using port 3128 instead of 443. After this, the attacker executed action A_{10} to run sslstop and sslport programs. The content of the concept A_{10} is vindicated by symptom S_{10}. The latter is indicated by the analysis of /etc/httpd/conf/httpd.conf which shows that the primitive HAVE_SSL was replaced by HAVE_SSS. The script sslstop modifies httpd.conf to disable the SSL support. sslport modifies httpd.conf to change the default SSL port (443) to something else (3128 in this case) and then to conceal any port scanner output that can provide the symptom informing about the use of SSL. A concealment relation is appended from Action A_{10} to symptom S_{14} in the IRPCM.

After installing the SucKIT rootkit, the attacker closed the first connection and reconnected to the same web server from 213.154.118.219. This action, namely A_{12}, succeeds the action A_{11} in the IRPCM, which consists in setting the parameters of /dev/ttyop, /dev/ttyoa and /dev/ ttyof to hide processes, addresses and files, respectively and then to conceal symptoms such as NetStat output. Action A_{11} conceals also the action of connecting to the server 213.154.118.219. A concealment relation is created from the concept A_{11} to the concept A_{12}.

Using the new shell, the attacker conducted three other actions. He first installed the swapd network sniffer (Action A_{14} in the IRPCM). This action is supported by the two symptoms S_{19} and S_{20}. S_{19} indicates that a trusted version of ifconfig showed that the network card was in promiscuous mode. S_{20} represents the output of a trusted version of the lsof command indicating that swapd was running using the pid 3153. Second, the attacker executed a remote FTP access (Action A_{16} in the IRPCM). This action is vindicated by the concept S_{18} representing the analysis of xinetd configuration. It shows that /etc/ ftpaccess is configured to allow anonymous access. Action A_{16} leads to the unauthorized U_5 showing that an unauthorized downloading of programs via FTP was performed. Third, the attacker installed psyBNC (Action A_{13} in the IRPCM) which is an Internet Relay Chat (IRC) proxy. This action conceals the unauthorized result U_4 which shows a communication with other crackers. By concealing U_4, the IRC program allows communications without revealing the intruder identity. Action A_{13} is vindicated by the symptom S_{17}. In fact the execution of a trusted lsof command on the compromised system shows that psyBNC is running. Using psyBNC, the attacker connected to the IRC server fairfax.va.us.undernet.org (Action A_{15} in the IRPCM). An edge labeled by a concealment relation is created from the concept A_{13} to the concept S_{17}.

6.3. Extracting Evidences from IRPCMs

We model the investigated system using six variables; namely *Pr*, *hhtplog*, *port*2003, *ConAddr*2003, *SorftLog*, and *AppSoft*. They represent the system privilege granted to the remote user (i.e., the attacker), the tail of the content of the web service http log file, the service running on port 2003, the IP address connected to port 2003, the content of residual software installation logs, and the additional software installed on the system.

The evidences extracted from the IRPCM in conjunction with the library of elementary actions are then used by the I-TLA logic to specify the set of potential attack scenarios. For the sake of space, we will only consider a specific part of the IRPCM and we will describe the related I-TLA specification. Concepts in the IRPCM having a degree of activation that exceeded a pre-defined threshold are traduced into I-TLA evidences.

The concept "Disk sectors: Log installation history" is traduced to history-based evidence in I-TLA. This evidence, which is provided by the log installation file, allows monitoring the content of variable *SoftInsLog*. The provided evidence represents an observation over such variable. Since it was targeted by an anti-investigation attack, it is equal to <-> showing that none log file, which could be left by the installed software, is on the system. Some concepts from the IRPCM, in the form of actions, are mapped to I-TLA actions. For instance, the two actions "Execution of mod_ssl/OpenSSL exploit" and "Install rk.tar.gz" are traduced to the two following I-TLA actions, say $A1$ and $A2$, respectively:

$$\begin{aligned} A1 \triangleq\ & \wedge Pr' = 1 \\ & \wedge httplog' = \langle modsslattack \rangle \\ & \wedge port2003 = "/bin/sh" \\ & \wedge \langle ConAddr2003, SoftInsLog, AppSoft \rangle \end{aligned} \tag{9}$$

$$\begin{aligned} A2 \triangleq\ & \wedge Pr = 2 \\ & \wedge port2003 = < - > \\ & \wedge ConAddr2003 = < - > \\ & \wedge \langle Pr, httplog, port2003, ConAddr, AppSoft \rangle \end{aligned} \tag{10}$$

Action $A1$ can be executed to compromise the web service using the SSL vulnerability. It consists in inducing the system to grant a shell on port 2003. Further to the execution of such action, an entry is appended to the HTTP log file showing that a suspicious behavior has occurred. Since, the exploitation of the OpenSSL vulnerability gives access using the privilege of the Apache user, variable *Pr* gets value 1. This value means that the access level is more privileged that the user access but

Copyright © 2009 SciRes.

less privileged than the root one.

Action *A*2 cannot be executed unless variable *Pr* is equal to 2 to mean that the user should have gained a root privilege on the system. The action consists in hiding the execution of services on port 2003, and the connection of suspicious hosts on port 2003, if the Linux commands ps and netstat are used. Actions *A*1 and *A*2 stand for timed event-based evidence showing that *A*1 is executed before *A*2 and both of them are part of the conducted attack scenario.

6.3.1. Executable Scenarios Generation by I-TLC

I-TLC was used to generate executable specification [18], of the potential attack scenarios from I-TLA specification. One potential attack scenario is generated and is shown by Figure 4. The scenario is composed of nine states where every state shows the value of the modeled variables. Edges linking states, are labeled by the name of the executed action.

The system starts with an empty HTTP and system log file. The attacker first connects to the web service and runs the modssl exploit to get system access with the Apache user privilege. After that, it connects to the shell granted on port 2003. Thus, variable *ConAddr*2003 gets the value of the IP address of the remote user. After that, the attacker makes a tentative to install the rk rootkit. The operation is logged to the installation log of this tool. However, since it fails, no files were integrated to the system directory and variable *AppSoft* remains unchanged. Later, the attacker conducts a storage-based anti-investigation attack to hide the content of the installation log file. It reconnects from another host identified by the IP address 213.154.118.218, escalates its privilege, installs the suckit rootkit, and configures it to hide the execution of the shell on port 2003 and connected the IP address.

In this execution, it can be noticed that the execution of action *A*2 creates state *s*3 which does not satisfy the content of the history-based evidence. In fact, since an anti-investigation attack was executed and detected during the IRPCM construction, I-TLC has allowed the execution of *A*2 because it modifies the content of variable *SoftInsLog* which was affected by the anti-investigation attack.

6.3.2. Hypothetical Actions Generation

I-TLC has generated some hypothetical actions. For the lack of space, we only kept one hypothesis among those generated. Starting from state s_6, I-TLC could not find an action described in I-TLA specification, which, if executed, lets obtaining a potential attack scenario that satisfies the history-based evidence. I-TLC looks within the

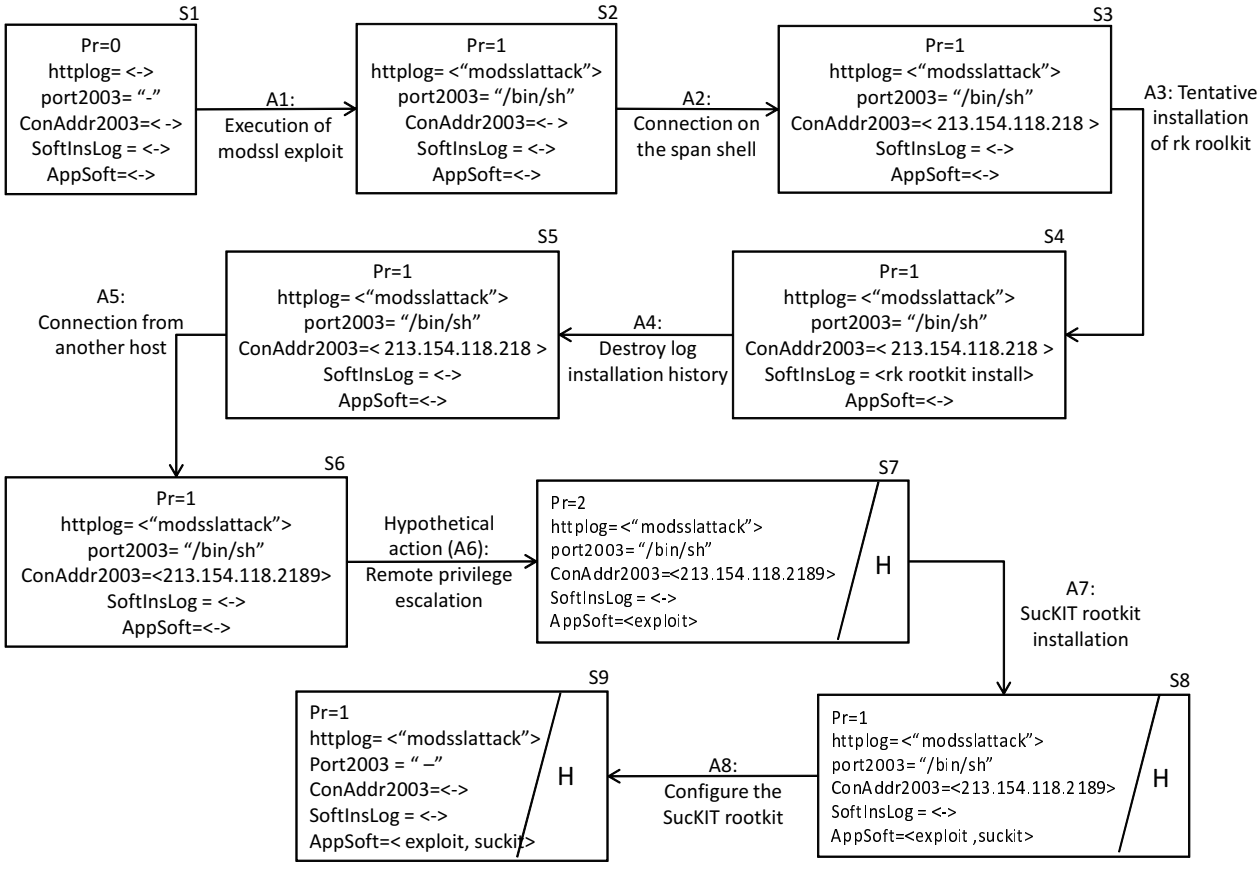

Figure 4. Fragment of the generated executable specification by I-TLC.

Copyright © 2009 SciRes.

library of hypotheses and generates, a, hypothetical action H, and executes it to move the system to state s_7. The hypothetical action consists in uploading an exploit to the compromised system and executing it to get a root privilege. Further to the execution of the hypothetical action, variable *pr* gets value 2, and the value *exploit* is appended to the content of variable *AppSoft*. Later actions $A7$ and $A8$ are executed. I-TLC specifies that states s_7, s_8 and s_9 are reachable under the hypothetical action H by setting their label equal to the singleton H.

7. Conclusions

In this paper, we have developed a system for digital investigation of networks security incidents. This system uses formal techniques as well as the IRT members' knowledge to analyze the attacks performed against the networks. We have introduced the intrusion response probabilistic cognitive maps that are constructed by the IRT upon the occurrence of the attack. A formal language has been introduced to help specifying the attack scenarios based on the cognitive map. A model checker was built to automatically extract the attack scenarios and a hypothetical concept is introduced here to help in the construction process. To illustrate the proposed system, we used it in a real case of security attack.

8. References

[1] P. D. Dixon, "An overview of computer forensics," IEEE Potentials, Vol. 24, No. 5, pp. 7–10, 2005.

[2] P. Stephenson, "Modeling of post-incident root cause analysis," International Journal of Digital Evidence, Vol. 2, No. 2, pp. 1–16, 2003.

[3] T. Stallard and K. Levitt, "Automated analysis for digital forensic science: Semantic integrity checking," Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, USA, 2003.

[4] P. Gladyshev, "Finite state machine analysis of a blackmail investigation," International Journal of Digital Evidence, Vol. 4, No. 1, 2005.

[5] P. Gladyshev and A. Patel, "Finite state machine approach to digital event reconstruction," Digital Investigation journal, Vol. 1, No. 2, pp. 130–149, 2004.

[6] B. D. Carrier and E. H. Spafford, "Categories of digital investigation analysis techniques based on the computer history model," Digital Investigation Journal, 3(S), pp. 121–130, 2006.

[7] S. Willassen, "Hypothesis-Based investigation of digital timestamps," Proceedings of Fourth Annual IFIP WG 11.9 International Conference on Digital Forensics, Kyoto, Japan, 2008.

[8] S. Y. Willassen, "Timestamp evidence correlation by model based clock hypothesis testing," Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia, 2008.

[9] A. R. Arasteha, M. Debbabi, A. Sakhaa, and M. Saleh, "Analyzing multiple logs for forensic evidence," Digital Investigation, Vol. 4, No. 1, pp. 82–91, 2007.

[10] A. Pal, H. T. Sencar, and N. Memon, "Detecting file fragmentation point using sequential hypothesis testing," Digital Investigation, Vol. 5, No. 1, pp. S2–S13, 2008.

[11] S. P. Peisert, "A model of forensic analysis using goal-oriented logging," PhD thesis, University of California, San Diego, 2007.

[12] A. S. Huff, "Mapping strategic thought," John Wiley & Sons, 1990.

[13] J. Krichene, M. Hamdi, and N. Boudriga, "Collective computer incident response using cognitive maps," IEEE International Conference on Systems, Man and Cybernetics, Hammamet, Tunisia, pp. 1080–1085, 2004.

[14] S. Rekhis, J. Krichene, and N. Boudriga, "Dig for net: Digital Forensic in networking," In Proceedings of the 3rd International Information Security Conference (SEC), Milan, Italy, 2008.

[15] B. D. Carrier and E. H. Spafford, "An event-based digital forensic investigation framework," Proceedings of Digital Forensic Research Workshop, 2004.

[16] B. Mangnes, "The use of Levenshtein distance in computer forensics," Master's thesis, Gjovik University College, 2005.

[17] E. Casey, "Digital evidence and computer crime," Second Edition, Academic Press, 2004.

[18] D. Drusinsky and J. L. Fobes, "Executable specifications: Language and applications," The journal of Defense Software Engineering, Vol. 17, No. 9, pp. 15–18, 2004.

[19] Y. Guan and A. K. Ghose, "Executable specifications for agent oriented conceptual modelling," Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), France, pp. 475–478, 2005.

[20] M. Hamdi, J. Krichene, and N. Boudriga, "Collective computer incident response using cognitive maps," Proceedings of IEEE conference on Systems, Man, and Cybernetics (IEEE SMC 2004), The Hargue, Netherland, 2004.

[21] S. Rekhis and N. Boudriga, "A formal approach for the reconstruction of potential attack scenarios," Proceedings of the International Conference on Information & Communication Technologies: From Theory to Applications (ICTTA), Damascus, Syria, 2008.

[22] F. Kröger and S. Merz, "Temporal logic and state systems," Springer, 2008.

[23] S. Rekhis and N. Boudriga, "Formal forensic investigation eluding disk-based anti-forensic attacks," Proceedings of Workshop on Information Security Applications, Jeju Island, Korea, 2005.

[24] S. Garfinkel, "Anti-forensics: Techniques, detection and countermeasures," Proceedings of the 2nd International Conference on I-Warfare and Security, Monterey, USA, 2007.

[25] G. C. Kessler, "Anti-forensics and the digital investigator," Proceedings of 5th Australian Digital Forensics Conference, Perth, Australia, 2007.

Copyright © 2009 SciRes.