DigForNet: Digital Forensic in Networking

Slim Rekhis, Jihene Krichene, and Noureddine Boudriga

Abstract Security incidents targeting information systems become more complex and sophisticated, and intruders might evade responsibility due to the lack of supporting evidences to convict them. In this paper, we develop a system for Digital Forensic in Networking (DigForNet) which is useful to analyze security incidents and explain the steps taken by the attackers. DigForNet uses intrusion response team knowledge and formal tools to reconstruct potential attack scenarios and show how the system behaved for every step in the scenario. The attack scenarios identification is automated and the hypothetical concept is introduced within DigForNet to alleviate lack of data related to missing evidences or investigator knowledge.

1 Introduction

Faced to the increase and sophistication of security incidents, security experts have started giving a great interest to the digital forensic investigation of security incidents. Defined in the literature as *preservation, identification, extraction, documentation and interpretation of computer data* [1], digital investigation aims to perform a post-incident examination of the compromised systems to identify conducted attack scenarios and attackers source, understand what occurred to prevent future similar incidents, and argument the results with non refutable proofs.

Performing a digital investigation is a challenging task. First, attacks may use multiple sources and become difficult to trace using available traceback techniques. Second, systems may not be initially prepared for investigation, leading to the absence of effective logs and alerts to be used for understanding the incident. In addition, the attackers may use a number of techniques to hide traces left on the compro-

Slim Rekhis[1], Jihene Krichene[2], and Noureddine Boudriga[3]
Communication Networks and Security Research Lab. University of the 7th of November at Carthage, Tunisia, e-mail: [1]slim.rekhis@isetcom.rnu.tn, [2]jkrichene@gmail.com, [3]nab@supcom.rnu.tn

mised system. Third, attack scenarios may use several automated tools that create intensive damaging activities. A large amount of data should thus be analyzed.

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: a) better filter the data to be analyzed and source of evidences to be explored, based on the skills developed by the IRT, and b) validate the results of the formal techniques by the IRT before presenting them and exploit them to accumulate knowledge about security incident. 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 providable 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 approaches, digital investigation of security incidents remains scarcely explored by these methods. Stephenson took interest in [8] to the root cause analysis of digital incidents and used Colored Petri Nets to model occurred events. The methodology may become insufficient if there is a lack of information on the compromised system that requires some hypotheses formulation. Stallard and Levitt proposed in [7] 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 provided in [2] a Finite State Machine (FSM) approach to the construction of potential attack scenarios discarding scenarios that disagree with the available evidences. However, if some system transitions (e.g., malicious event) are unknown, the event construction may freeze.

We develop in this paper, a system for Digital Forensic in Networking (DigForNet). It integrates the analysis performed by the IRT on a compromised system, through the use of the Incident Response Probabilistic Cognitive Maps (IRPCMs). DigForNet provides a formal approach to identify potential attack scenarios using I-TLA logic. The latter allows to specify different forms of evidences, and identify an attack scenario as a series of elementary actions retrieved from a used library, that, if executed sequentially on the investigated system, would produce the set of available evidences. To develop the concept of executable attack scenarios showing with details how an attack is performed progressively on the system, DigForNet uses I-TLC, an automated verification tool for I-TLA specifications. To handle unknown attacks, DigForNet integrates a technique for generating hypothetical actions to be appended to the scenario under construction.

DigForNet contribution is three-fold. First, to the best of our knowledge, it is the first investigation system that reconciles in the same framework conclusions derived by the incident response team and theoretical and empirical knowledge of digital investigators. Second, using the concept of hypothetical actions, DigForNet stands out from the other existing approaches and allows to generate sophisticated and unknown attack scenarios. Third, most of the techniques brought by DigForNet can be automated which makes it a promising computer-assisted investigation tool.

This paper is organized as follows. Section 2 describes the DigForNet's methodology for reasoning about security incidents. The use of the IRPCM technique is described in Section 3. Section 4 describes I-TLA as a 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 introduces I-TLC showing how it can be used to generate executable attack scenarios. Section 6 illustrates with an example the use of DigForNet in investigating a real security incident. Finally, Section 7 concludes the paper.

2 Methodology of structured investigation

DigForNet methodology is composed of five steps in a waterfall model as shown in Figure 1. The first step collects evidences available within three different sources, namely the operating systems, networks, and storage systems. DigForNet integrates the incident response team contributions under the form of Incident Response Probabilistic Cognitive Maps (IRPCMs). An IRPCM is nothing but a directed graph representing security events, actions and their results. It is built during the second step 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.

The third step generates a formal specification. Sets of evidences and actions are extracted from the cognitive map for the formal specification of the potential attack scenarios. A formal approach is necessary for this purpose. DigForNet uses a 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 elementary attacks, and the initial system state. In 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 fourth step generates of executable potential attack scenarios using a model checker tool associated with the formal specification. DigForNet uses Investigation-based Temporal Logic Model Checker called I-TLC. The latter rebuild the attack scenarios in forward and backward chaining processing, showing details of all intermediate system states through which the system progresses during the attack. I-TLC 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 fifth 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 to check the hypotheses added by I-TLC and update the initial IR-PCM where concepts can be omitted because they do not present an interest for the attack scenario construction, while other concepts corresponding to the hypothetical actions can be added to the IRPCM and linked to the other concepts. Links in the IRPCM are deleted in the case where the concepts at their origin or end are omitted. Hypothetical actions are also added to the attack library. In addition, tools collecting the evidences are enhanced to detect the newly discovered vulnerabilities.

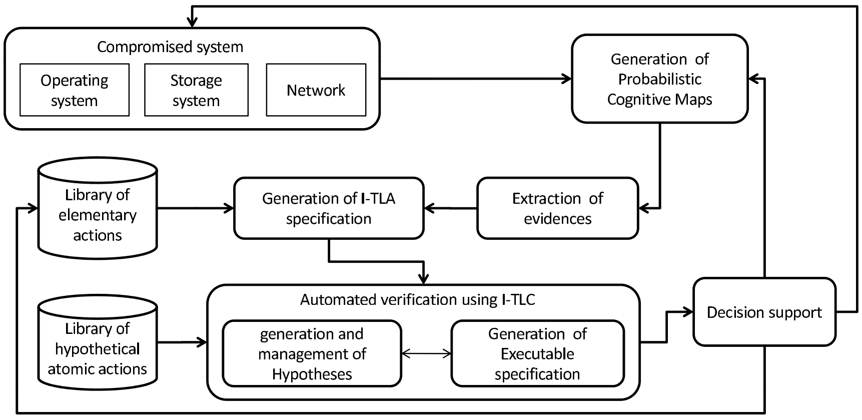
Compromised system
Operating system
Storage system
Network
Generation of Probabilistic Cognitive Maps
Library of elementary actions
Generation of I-TLA specification
Extraction of evidences
Automated verification using I-TLC
Library of hypothetical atomic actions
generation and management of Hypotheses
Generation of Executable specification
Decision support

Fig. 1 DigForNet Methodology

3 Intrusion Response Probabilistic Causal Maps

We have studied in [3] a new category of cognitive maps to support intrusion response. 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. IRPCMs provide a foundation to investigate and explain security attacks which have occurred in the past and predict future security attacks. 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 conflict between them.

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. In this graph, the nodes represent concepts belonging to the network security field, while the edges represent 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 (e.g., system crashes, existence of new user accounts or files). An action is a step taken by a user or a process in order to achieve a result (e.g., probes, scans, floods). An unauthorized result is an unauthorized consequence of an event (defined by an action directed to a target, e.g., increased access, disclosure of information). IRPCM concepts are labeled by values in the interval $[0, 1]$ informing about the activation of the correspondent concepts.

IRPCM edges link concepts to each others. Each 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 relationship between the two nodes (examples include $<_t$, I/O, CE) and q_{ij} (taking values in $]0, 1]$) is the probability expressing the certitude degree that the relationship π_{ij} really exists between concepts c_i and c_j. Quantitative values are given by security experts. Notice that the predicate π_{ij} depends on the nature of the concepts c_i and c_j. For the reason of simplicity, we consider four cases in this paper:

1. c_i is a symptom and c_j is a symptom or an action: π_{ij} expresses an input/output relationship ($\pi_{ij} = I/O$). Part of output of c_i is the input of c_j.
2. c_i and c_j are two actions: π_{ij} expresses a temporal relationship between the two concepts ($\pi_{ij} =<_t$). c_i is an action that precedes c_j.
3. c_i is an action and c_j is an unauthorized result: π_{ij} expresses the causality existing between the action and the unauthorized result ($\pi_{ij} = CE$).
4. c_i and c_j are the same concept: π_{ij} is the identity.

3.2 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 network services, or any sign indicating the occurrence of a malicious action 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 seven steps:

1. Collect 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 collected events. Events which do not belong to the previous IRPCM are added. Links related to the newly considered concepts are also added to the IRPCM.
5. Refine the IRPCM by omitting the nodes that the IRT members find not interesting for the investigation activity.
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.

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.

3.3 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 to assign activation degrees to the concepts as follows:

$$\begin{aligned} dac\colon\ C &\rightarrow [0,1] \\ c &\mapsto dac(c) \end{aligned}$$

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

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. $G = \{c \in C/\exists d \in F, (d,c) is\,a\,relation\}$. Then, $dac(c) = sup_{d\, \in\, G}\{q_{dc}dac(d)\}$.
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 construct the IRPCM corresponding to this situation. The concepts that represent the collected events are activated and will form the set I. 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 nodes having a degree greater than a predefined threshold. These nodes will be used as evidences for the formal specification.

4 Generation of a formal specification of attack scenarios

The Investigation-based Temporal Logic of Actions, I-TLA [6], is a logic for the investigation of security incidents. It is an extension to S-TLA logic [5], which is itself an extension to the TLA logic [4]. 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 model and specify available set of evidences, and generate a specification describing potential attack scenarios (as a series of elementary actions extracted from a library describing legitimate and malicious events) that satisfy these 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 [6] for a complete understanding 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 that 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). This form of representation allows a security expert to observe how its system progresses during the attack and how it interacts with the actions executed in the scenario. I-TLA supports four different forms of evidences, namely history-based, non-timed events-based, timed events-based, and predicate-based evidences.

- **History-based evidences:** 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 provide evidences as histories of the value of the monitored system components, during the spread of an attack scenario. $Obs(\omega)$ is obtained as follows:

  1. Transform very state s_i to $\hat{s}_i$ using a labeling function that makes the value of every variable v in s_i be: invisible (in that case it will be represented by ε), equal to a fictive value, or unmodified.
  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 $Obs(\omega) = E$.
- **Ordering of observations:** As the scope of observations differs, they may not allow to notice that the system has progressed during the attack at the same time. I-TLA allows to specify for two given history-based evidences, which one is expected to vary first/last when the attack scenario starts/finishes. Consider the following example involving an attack scenario ω, and two history-based evidences $OBS = [e_1, ..., e_n]$ and $OBS' = [e_1^{'}, ..., e_m^{'}]$, generated by observation functions $Obs()$ and $Obs'()$, respectively. OBS allows to notice the occurrence of an incident before OBS', if and only if: $\exists \omega_x$ such that: $\omega = \omega_x \omega_y \wedge Obs(\omega_x) = [e_1, ..., e_j] \wedge Obs'(\omega_x) = e_1^{'}$ for some j $(1 < j \leq m$.
- **Non-timed events based evidences:** Constructed attack scenarios may differ by the manner in which observations are stretched and stuck 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, that specify the modification pattern of variables values through an execution. The following evidence E, for instance, states that predicate p_1 switches to value true in the same state the predicate p_2 switches to value false ($E \triangleq \forall \langle s_i, s_{i+1} \rangle \in \omega : (s_i \nvDash p_1 \wedge s_{i+1} \vDash p_1) \Rightarrow s_i \vDash p_2 \wedge s_{i+1} \nvDash p_2)$). Taking into consideration the availability of a non-timed event-based evidence E, consists in generating, an attack scenario ω such that $\omega \vDash E$.
- **Timed events-based evidences:** Starting from a set of available alerts, an investigator can extract some indications related to occurred events. I-TLA defines a 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 without requiring 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: $(A_x(s_i, s_{i+1}) = true \wedge A_{x+1}(s_j, s_{j+1}) = true$ for some $j \geq i + 1)$.
- **Predicate-based evidences:** An unexpected system property, is a preliminary argument supporting the incident occurrence (e.g., the integrity of a file was violated). I-TLA defines a predicate-based evidence as a predicate, say E, over system states, that characterizes the system compromise. An execution ω satis-

fies evidence E, if E divides ω into two successive execution fragments ω_1 and ω_2. ω_1 is composed of secure states ($\forall s \in \omega_1$: $s \nvDash E$), while ω_2 is composed of insecure system states ($\forall s \in \omega_2$: $s \nvDash E$).

4.2 Illustrative example

We consider a system under investigation which is specified by three variables x, y, and z. The initial system system state, described in advance, states that 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: $A_1 \triangleq (x' = x) \wedge (y' = y+1) \wedge (z' = z+2)$ and $A_2 \triangleq (x' = x+1) \wedge (y' = y) \wedge (z' = z/2)$.

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 first two represent history-based evidences, defined as $E_1{=}\langle 0\varepsilon\varepsilon, 1\varepsilon\varepsilon, 2\varepsilon\varepsilon\rangle$ and $E_2{=}\langle \varepsilon 0\varepsilon, \varepsilon 1\varepsilon, \varepsilon 2\varepsilon, \varepsilon 3\varepsilon\rangle$. They 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 applied to a state s, it 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 $p_1 \triangleq x = 1$, which is false in a state s_i, could not switch to true in the next state s_{i+1}, unless predicate $p_2 \triangleq z \neq 4$ is true in that state. Finally, evidence E_5, indicates that sequence of events (A_1, A_2) is part of the attack scenario.

Figure 2 shows how I-TLA guarantees the satisfaction of evidences during construction of the potential attacks. Two potential attack scenario satisfying the available evidences are provided by I-LA, namely ω_1 and ω_2. The first scenario ω_1 is described as $\omega_1 = \langle s_1, s_3, s_4, s_7, s_{12}, s_{15}\rangle$, 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 $\omega_1 = \langle s_1, s_3, s_5, s_9, s_{11}, s_{18}\rangle$.

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 $\langle s_0, s_1\rangle$ is observed by $Obs_1()$ as $\langle 0\varepsilon\varepsilon, 1\varepsilon\varepsilon\rangle$ and by $Obs_2()$ as $\langle \varepsilon 0\varepsilon\rangle$. The event A_2 is thus 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 became already true in state s_3 and should not change again to false in state s_8. 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.

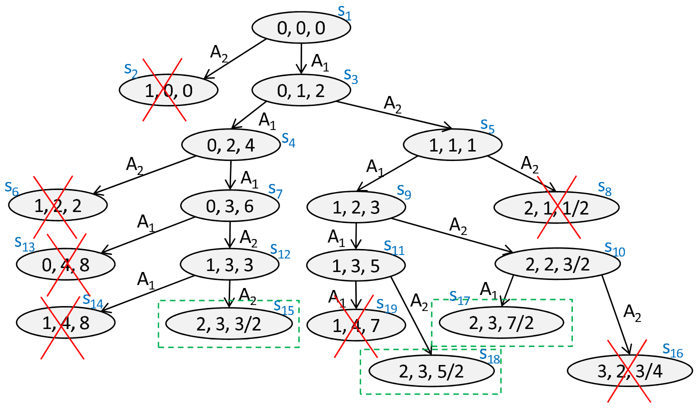

Fig. 2 I-TLA attack scenario generation: an illustrative example

4.3 From IRPCM to I-TLA specification

Starting from the IRPCM built by the IRT, useful information, 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 alert files indicate the occurrence of events whose position in the constructed attack scenario cannot be determined. They will typically be transformed to a 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 identifies the system variable affected by the unauthorized consequence and then uses it to describe the evidence.

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 [6], a model checker for I-TLA$^+$ specifications can be used. I-TLC is somehow an extension to TLC, the model checker of TLA$^+$ specification.

I-TLC represents a node in the graph as a tuple of two information: *node core* and *node label*. The node core 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.

As the generation of potential attack scenarios may fail if the library of actions is incomplete, 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) is available. This detail involves a description of how these internal system components are expected to behave (if an atomic actions is executed on them) and how they depend on each other. These internal system components are modeled by a specific set of variables 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 a pair of states $\langle s, t \rangle$. 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 sequences 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.

To generate potential scenarios of attacks, DigForNet uses I-TLC Model Checker, which follows three phases. The reader is referred to [6] for a detailed description of I-TLC algorithms.

1. Initialization phase: During this step, the generated scenarios graph is initialized to empty, and each state satisfying the initial system predicate is computed, appended to the graph with a pointer to the $null$ state, and a label equal to $\emptyset$ (as no hypothetical action is generated).

2. Forward chaining phase: The algorithm starts from the set of initial system states, and computes in forward chaining manner all the 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 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 a sound, consistent, complete, and minimal the set of hypothetical actions under which the new system state is reachable.

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. The new generated predecessor 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 allows to alleviate this problem.

6 Case study

We concentrate on the following case study, related to the investigation of a DoS attack against a web server. Upon the occurrence of the incident, DigForNet collects traces from the network IDS, the web server log files, and the storage-based IDS located in the web server.

IRPCM generation

The IRT members analyze the compromised system and the output of the security tools to build the IRPCM. The generated graph is represented in Figure 3 and is composed of six symptoms, seven actions, and two unauthorized results. They appear in by dashed, continuous, and dotted ellipses, respectively. The first steps of the construction process are described as follows. Snort IDS provided two alerts related to the occurrence of a buffer overflow attack and a reconnaissance attack. The web server log file indicates that a malformed URL was used by some users. These alerts form the symptoms linked to the action "probe the web server version". This action precedes the action "Launch a remote buffer overflow on HTTP service". The occurrence of the latter is vindicated by the alert provided by the network monitoring tool. The remote buffer overflow action is used to execute some privileged commands aiming at sending network traffic to an unused port.

Having appended the set of concepts, IRT members define the edges linking the nodes and set their labels. We highlight here the existence of some high probabilities related to edges linking the concepts $S4$ to $A2$, $S5$ to $A6$, and $A6$ to $U2$. The degree of activation of the concepts $S4$, $S5$, $A1$, $A6$ and $U2$ are set to 1 as their existence is well vindicated by the content of the log and alert files provided by the web server, the network monitoring tools, and the NIDSs.

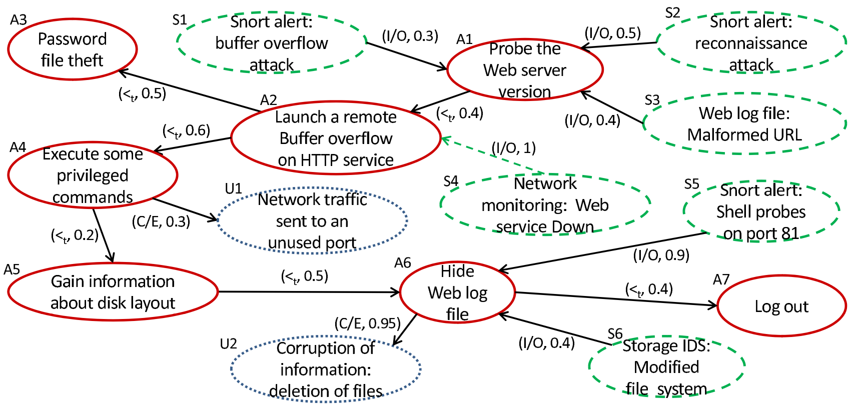
A3
Password file theft
S1
Snort alert: buffer overflow attack
(I/O, 0.3)
A1
Probe the Web server version
(I/O, 0.5)
S2
Snort alert: reconnaissance attack
(<t, 0.5)
A2
Launch a remote Buffer overflow on HTTP service
(<t, 0.4)
(I/O, 0.4)
S3
Web log file: Malformed URL
(<t, 0.6)
(I/O, 1)
A4
Execute some privileged commands
U1
Network traffic sent to an unused port
S4
Network monitoring: Web service Down
S5
Snort alert: Shell probes on port 81
(C/E, 0.3)
(<t, 0.2)
(I/O, 0.9)
A5
Gain information about disk layout
(<t, 0.5)
A6
Hide Web log file
(<t, 0.4)
A7
Log out
(C/E, 0.95)
U2
Corruption of information: deletion of files
(I/O, 0.4)
S6
Storage IDS: Modified file system

Fig. 3 Case study: Generation of the IRPCM

Extracting evidences from IRPCMs

To extract useful information from the IRPCM, the IRT members defined an extraction threshold equal to 0.9. Concepts in the IRPCM having a degree of activation value that exceeds 0.9 are retained to be translated into I-TLA evidences. The investigated system is modeled using four variables, namely Pr, $Srv80$, $Weblog$, $Srv81$. They represent the privilege granted to the remote user, the service granted on port 80, the tail of the content related to the web service log, and the service granted on port 81. Symptoms $S4$ and $S5$ are traduced to a history-based evidence described in I-TLA in the form of $\langle \varepsilon\text{"}http\text{"}\varepsilon\varepsilon,\ \varepsilon\text{"}noservice\text{"}\varepsilon\varepsilon\rangle$, and $\langle \varepsilon\varepsilon\varepsilon\text{"}noservice\text{"},\ \varepsilon\varepsilon\varepsilon\text{"}/bin/sh\text{"}\rangle$, respectively. The first evidence is generated by the network monitoring tool that allows to monitor variable $srv80$. The second evidence is generated by the NIDS that verifies that none traffic is directed to unused server ports. In other words, it monitors variable $srv81$. The unauthorized result $U2$ is traduced to a predicate-based evidence in the form of $weblog = $ "_". Each one of actions $A1$ and $A6$ is mapped to a single action in the I-TLA library. These actions are described as follows:

$$\begin{array}{ll} Act1 \triangleq \wedge Pr = 0 \wedge Srv80 = \text{"}http\text{"} & \\ \quad \wedge Pr' = 1 \wedge Weblog' = \text{"}get/?\text{"} & Act3 \triangleq \wedge Pr \geq 2 \wedge Weblog' = \text{"\_"} \\ \quad \wedge \text{UNCHANGED } \langle Srv81, Srv80 \rangle & \quad \wedge \text{UNCHANGED } \langle Pr, Srv81, Srv80 \rangle \end{array}$$

Action $A1$, for instance, cannot be executed unless Pr and $Srv80$ values are set to 0 and "$http$", respectively. Once executed, it sets the value of Pr equal to 1 (a non privileged access is granted), and the value of $Weblog$ equal to "$get/?$" (the tail of the web log file indicates that a specific URL was requested to probe the web server version). Actions $Act1$ and $Act3$ form a timed event-based evidence indicating that sequence $(Act1, Act3)$ is part of the attack scenario. 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.

Executable scenarios generation by I-TLC

Starting from the I-TLA$^+$ specification, I-TLC generates one potential executable attack scenario, composed of six states, described in Figure 4. Every state in the scenario describes the value of the four system variables used in the I-TLA$^+$ specification. Edges linking states, are labeled by the name of the executed I-TLA action. An attacker gets an unprivileged access to the web server and requests a specific URL that probes the web server version. After that, it conducts a buffer overflow attack, by exploiting a remote vulnerability in the web service. The web service becomes down and the intruder escalates its privilege on the compromised system. In the third step, the intruder executes an anti-investigation attack on the file system, hiding the content of the blocks related to the web log file.

I-TLC has generated some hypothetical actions. For the lack of space, we only kept one hypothesis among those generated. Starting from state s_4, I-TLC could not find in I-TLA specification an action which can be executed. It looks within the library of hypotheses if it is possible to generate a hypothetical action, which if executed, will let the system progress to a state that satisfies all available evidences. I-TLC generates a hypothetical action H and executes it to move the system to state s_6. The hypothetical action consists in attaching a shell to port 81 to be used by a backdoor allowing the intruder to maintain his access to the system. Later the intruder logs out. I-TLC specifies that states s_6 and s_7 are reachable under the hypothetical action H by setting their label equal to the singleton $\{H\}$.

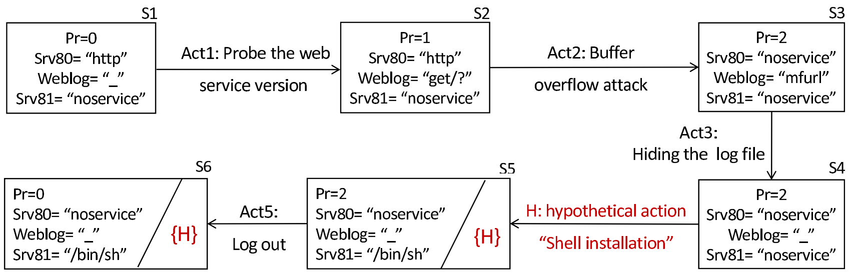

Fig. 4 Executable scenarios generation by I-TLC

7 Conclusion

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.

References

1. P.D. Dixon. An overview of computer forensics. *IEEE Potentials*, 24(5):7–10, dec 2005.
2. Pavel Gladyshev. Finite State Machine Analysis of a Blackmail Investigation. *International Journal of Digital Evidence*, 4(1), May 2005.
3. Mohamed Hamdi, Jihene Krichene, and Noureddine BOUDRIGA. Collective computer incident response using cognitive maps. In *Proceedings of IEEE conference on Systems, Man, and Cybernetics (IEEE SMC 2004)*, The Hargue, Netherland, October 10-13 2004.
4. Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
5. Slim REKHIS and Noureddine BOUDRIGA. A temporal logic-based model for forensic investigation in networked system security. In LNCS 3685 Springer Verlag, editor, *Third International Workshop Mathematical Methods, Models and Architectures for Computer Networks Security*, pages 325–338, St. Petersburg, Russia, September 2005.
6. Slim REKHIS and Noureddine BOUDRIGA. A formal approach for the reconstruction of potential attack scenarios. In *Proceedings of the International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA)*, Damascus, Syria, April 2008.
7. Tye Stallard and Karl Levitt. Automated analysis for digital forensic science: Semantic integrity checking. In *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, Nevada, USA, December 2003.
8. Peter Stephenson. Modeling of post-incident root cause analysis. *International Journal of Digital Evidence*, 2(2):1–16, 2003.