



CNAS REPORT

Printed January 2008

CNAS-2008-002
Public

Supersedes CNAS-2007-002
Dated Dec. 2007

A Temporal Logic-Based Model for Forensic Investigation in Networked System Security

Slim Rekhis
Noureddine Boudriga

Prepared by
CN&S Research Lab.

The Communication Network and Security (CN&S) research Laboratory,
(Created in 1999, 02/UR/11-08) is located at the Communication School of Engineering
(University of 7th of November at Carthage, Tunisia).

Approved for public release.

Copyright © 2007 by the Communication Networks and Security Research Lab. All rights reserved.

NOTICE: No part of this publication may be reproduced, stored in a retrieval system, or transmitted without written authorization from the CN&S research lab.

Available from

CN&S research lab.
Engineering school of communications.
Techno-parc El Ghazala, Route de Raoued.
Ariana, 2083, Tunisia.

Telephone: (+216) 71857000 (ext. 2104)
Facsimile: (+216) 71856829
E-Mail: cnas@laposte.net

Approved for public release

Professor Nouredine Boudriga
Head of CN&S research lab.

A Temporal Logic-Based Model for Forensic Investigation in Networked System Security

Slim Rekhis and Nouredine Boudriga

CN&S Research Lab., University of the 7th Of November at Carthage, Tunisia
smr@certification.tn
nab@supcom.rnu.tn

Abstract. Research in computer and network forensic investigation has recently addressed the development of procedural guidelines, technical documents, and semi-automation tools. It has however omitted the need of formal proof. This work provides a novel approach that formalizes and automates the proof in digital forensic investigation. First, it brings out a formal logic-based language, called S-TLA⁺, to enable reasoning on systems with uncertainty, by adding forward hypotheses to fulfill potential lack of details. S-TLA⁺ is suitable for the description of evidences, as well as elementary scenarios fragments representing the investigators knowledge. Secondly, the proposal provides an automated verification tool, S-TLC, to prove the correctness of S-TLA⁺ specifications. It checks whether there are possible hacking scenarios that meet the available digital evidences, and explores additional evidences. To demonstrate its effectiveness, the formalized analysis is applied on a compromised host.

1 Introduction

The growth of the number of digital security incidents and the sophistication of the intrusions techniques made it impossible to completely prevent attacks. Therefore, it becomes necessary to react efficiently to security incidents. Computer forensic investigation, defined as “preservation, identification, extraction, documentation and interpretation of computer data” [1], enables achieving these objectives while performing a post-incident examination: a) evidence collection; b) attack scenarios and relating security weakness determination; and c) result argumentation with methods and techniques that are well-tested and proved.

During the recent years, the literature has addressed two main themes: a) contribution to the development of technical documents specific to the investigation of various operating systems and b) writing of procedural guidelines for forensic investigation. It has omitted any need of formalization and proof automation in digital forensic investigation, reducing consequently the results accuracy, and analysis practicality. Formalization allows an explicit and unambiguous representation of forensic investigator’s knowledge and observations. The proof automation makes the generated investigation deductions relevant even with a huge amount of data. It lets investigators argue about complex scenarios without a need for advanced skills, nor a priori knowledge about the incident causes.

Formal digital forensic investigation has interested few works that differ according to the techniques and methodologies they used. [2] used Colored Petri Nets to model digital postmortems investigation as a time-line of events. It focused on determining the set of causes that enabled the security incident to success, so that the appropriate countermeasures can be foreseen. Nevertheless, the methodology does not model incident effects, and does not support hypotheses formulation when details are missing. [3] presented an automated diagnosis system that generates possible attacks sequences using a plan recognition technique, simulates them on the victim model, and performs pattern matching recognition between their side effects and log files entries. This technique assumes that attack activity is logged, which is in contradiction with the fact that complex attack scenarios may subvert logging daemon and alter logs before hackers leave the system. [4] used an expert system with a decision tree to search through evidences for potential violations of invariant relationship between digital objects. The methodology is useful in reducing the amount of data to be analyzed. Nevertheless, it roughly depends on the system time granularity and the degree of preciseness that the system uses to record time on objects.

This paper extends the work of [5]. First, it brings out a new logic-based language, entitled S-TLA⁺. Using a temporal logic of security actions, it offers an important enhancement of the formal specification language TLA⁺[6]. S-TLA⁺ is founded on a logic formalism that let adding forward hypotheses whenever there is lack of details (information may be corrupted by hackers) to understand the system. Second, the proposal is completed with an automated verification tool, called S-TLC, to prove the correctness of S-TLA⁺ specifications. The tool is based on the enhancement of the TLC model checker [6,7]. It is fitted to the automated diagnosis of digital security incidents.

Our contribution is straightforward. First, the proposed approach is completely independent from any computer security technology or incident. It allows arguing about sophisticated hacking scenarios as it tolerates potential lack of details. Second, S-TLA⁺ takes advantage from the richness of the formal specification language TLA⁺ to support advanced description of scenarios and evidences, namely using temporal modalities.

The remaining of this paper is organized as follows: First, the forensic investigation model is defined in Section 2. Next, Section 3 defines the novel S-TLA logic and emphasizes on the new concepts and modifications added to TLA. In Section 4, the formal specification language S-TLA⁺ is defined and demonstrated how it can be used within forensic investigation. Section 5 presents S-TLC, explains how it represents states, and describes how hacking scenarios are inferred both using forward and backward chaining. In Section 6, the proposal is exemplified by a case study. Finally, the work is concluded in Section 7.

2 The Computer Forensic Investigation Model

Given a set of evidences collected further to the occurrence of a security incidents, the proposal aims to first reconstruct to potential hacking scenarios where

hypotheses are advanced whenever needed, and secondly, provides any additional evidences. Alike [8], we consider a hacking scenario as a combination of more generic and reusable fragments, which are basically described in advance without an a priori knowledge about the whole hacking scenario that is looked for. Every scenario fragment is depicted by an optional set of hypotheses underlying the scenario-fragment occurrence, a set of pre-conditions that must be satisfied, and a set of actions to achieve a sub-goal of the whole scenario objective. The inclusion of hypotheses is due to the fact that investigation on sophisticated attack scenarios needs to be tolerant to potential lack of data. The latter is generated by intruders who want to alter any trace that could prove their identity or activity.

As the combination of scenario fragments leads to the accumulation of hypotheses, care need to be taken from inconsistency introduction. In fact, some hypotheses are contradictory with each other and could not arise in the same whole hacking scenario. Moreover as hypotheses are described by a set of relations between variables and values, two hypotheses using the same variable with different values might make no sense if grouped together in a scenario.

Figure 1 shows a set of attack scenarios relative to an unauthorized modification of access accounts on a remote server. The attack can be achieved after: 1) exploiting a remote vulnerability that grants privileged access; 2) escalating one’s privilege via local vulnerability exploit, 3) Logging to the system from a trusted server. The node Log from a trusted server X is composed by a hypothesis stating that a trust relationship is established between servers S and X , a post-condition stating that the user U_{sr} is being logged to the server X at that time and an action asserting a telnet connection by the user to the server S .

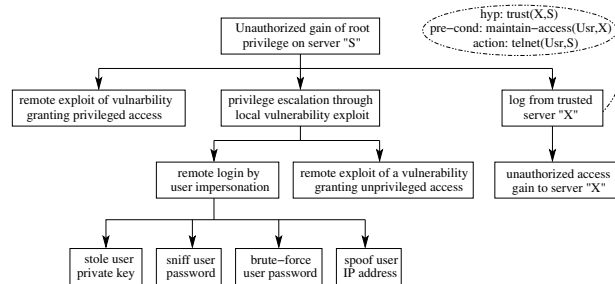


Fig. 1. Attack scenarios model

3 S-TLA: An Extension to the Temporal Logic of Actions

We provide in the following a Temporal Logic of Security Actions, S-TLA, as an extension to the Temporal Logic of Actions, TLA. We emphasize only on the new introduced concepts regarding TLA, as S-TLA embodies TLA and a TLA specification is indeed an S-TLA specification. TLA was introduced by Leslie Lamport for the specification of distributed and asynchronous systems [9].

Suppose, for instance, a formal system description that should involve a detail (value progress) of its n dependent variables, but some of them are unknown. To overcome such lack of details, it is conceivable to use a formalism that let enunciate hypotheses whenever needed. As denoted by Figure 2, we want to make TLA able to describe a system progress from a state s to a state t , further to the execution of an action \mathcal{A} and under a hypothesis $H_{\mathcal{A}}$.

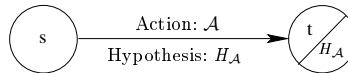


Fig. 2. State transitions under hypotheses

S-TLA Constrained Variables: We introduce a new set of variables, called constrained variables set V_C , to encompass the variables representing hypotheses. V_c is disjoint from sets V_F and V_R that represent flexible and rigid variables sets, respectively. Such separation is of great importance since we are looking during verification phase (c.f. section 5.1) to reach a given system state under a minimal set of hypotheses. Moreover as a hypothesis, once enunciated, might affect the system behavior, we assume that a constrained variable, whose value once set during a system state transition, could not be valued differently afterwards all through the system behavior.

S-TLA State: As in TLA, a state remains a valuation of all system variables. Precisely, it is an assignment from the collection Val of values to the set $Var = V_F \cup V_R \cup V_C$. A state can thus give information on the set of enunciated hypotheses that let it being reachable from the initial system state.

S-TLA Fictive Value ∇ : As a state is a valuation of all variables, a constrained variable should have a value even if there is no enunciated hypothesis yet. To bridge this gap, a new fictive S-TLA value described by the symbol ∇ is introduced to represent the value of a constrained variable that up to the moment was not used to make a hypothesis. Broadly, a state with a constrained variable whose value is different from ∇ means that there is an enunciated hypothesis to reach the related state.

S-TLA Assumption Operator $''$: We introduce a new S-TLA operator entitled assumption operator $''$ to denote the value of a constrained variable in the new state. This operator is different from the TLA prime operator. It changes the value of a constrained variable only if its value is different from ∇ . We define assumed and non-assumed variables to refer respectively to new and old state of constrained variables. In this way, we let $V_A \triangleq \{x'' \mid x \in V_C\}$ be the set of assumed variables.

S-TLA Inconsistency: We define an S-TLA inconsistency as a predicate containing constrained variables, constants, and constants operators [9]. Informally, an inconsistency denotes a combination of hypotheses that must not be observed inside a system behavior. Semantically it is true or false for a state. If it is true for a state t , then the system transition on the way to that state should not be followed. Hereinafter, we denote an S-TLA inconsistency using the symbol \perp .

S-TLA Action and Hypothesis: An S-TLA action is a conjunct between two expressions. The former is optional, of type boolean, denotes some hypotheses, and contains assumed and non-assumed variables. The latter is the old TLA action containing primed and unprimed variables. Semantically, given an inconsistency \perp , an S-TLA action \mathcal{A} is true for a pair of states $\langle s, t \rangle$ iff,

- $\mathcal{A}(\forall v \in V_F : s(v)/v, t(v)/v') = true$: By replacing each unprimed flexible variable in action \mathcal{A} by $s(v)$; the value of v in state s , and each primed flexible one by $t(v)$, the boolean resultant expression equals true.
- $\mathcal{A}(\forall v \in V_C : s(v)/v, t(v)/v'') = true$: By replacing each non-assumed constrained variable v in the action \mathcal{A} by $s(v)$ and each assumed constrained one v'' by $t(v)$, the boolean resultant expression equals true.
- $\forall v \in V_c / s(v) \neq \nabla : s(v)/v = t(v)/v$: The set of constrained variables whose values have been stated by a hypothesis (e.g. different from ∇) somewhere before, retain the same value in state s and t .
- $\perp(t) = false$: The predicate \perp must not hold in the state t , that is $(t \not\models \perp)$.

S-TLA Specification Formula: We introduce the predicate $IsTrue_{\mathcal{A}}(\perp)$ to be equal true if and only if \perp is true further to the execution of action \mathcal{A} . We define $NI_v(\mathcal{N}, \perp)$, *No Inconsistency* on action \mathcal{N} as: $NI_v(\mathcal{N}, \perp) \equiv \text{ENABLED } \mathcal{N} \wedge \neg IsTrue_{\mathcal{N}}(\perp) \Rightarrow \langle \mathcal{N} \rangle_v$ to states: if action \mathcal{N} is enabled and if its execution does not let inconsistency \perp equal true, then action \mathcal{N} occurs. We define ϕ as the system specification formula that generates an infinite behavior $\mathfrak{h} = \langle s_0, s_1, s_2, \dots \rangle$ (denoting the system progression) where no inconsistency \perp is holding in any state $s_i \in \mathfrak{h}$. The resultant form is as follows: $\phi \triangleq \exists x : Init \wedge \square[\mathcal{N}]_v \wedge L \wedge NI_v(\mathcal{N}, \perp)$. Except the quoted syntactically and semantically modifications, the remaining TLA notions including Fairness, stuttering, and temporal modalities are preserved.

4 S-TLA⁺: A Formal Language for Writing Specifications

We define S-TLA⁺ as a language for writing specifications in S-TLA, it embodies TLA⁺ [6] with some add-ons in the module structure (the lowest granular part of a TLA⁺ specification,) and in the constant and non constant operators. TLA⁺ is the high-level specification language that is based on TLA, and extended by notations of set theory (Zermelo Fraenkel set theory) and syntactic structuring mechanisms. To describe S-TLA⁺, we concentrate only on the introduced modifications as outlined hereinafter:

a) Module-Level constructs: The expression `CVARIABLES v_1, \dots, v_n` adds the declaration of constrained variables, distinguishing them from non-constrained ones, which remain declarable using `VARIABLES` statement.

b) Non constant S-TLA⁺ operators: Given a constrained variable h , we denote by h'' the value of h in the next state. Moreover, `UNTOUCHED h` replaces the expression $h'' = h$

c) Constant S-TLA⁺ operators: we denote by ∇ a fictive value to represent the constrained variable value, before a hypothesis is enunciated.

4.1 Standard Form of a S-TLA⁺ Specification

The first part of Figure 3, [5], illustrates a typical S-TLA⁺ specification, described by module *SpecExpl*. The specified system is described by formula *spec*, while the initial system state is described by predicate *Init* (no hypotheses are enunciated as constrained variables g and h are both equal to ∇). Action *A*, for instance, is true for a pair of states $\langle s, t \rangle$ if (1) the value that t assigns to x is 1 higher than the value that s affects to x , (2) under the hypothesis $g'' = 1$, and (3) without t being reached under the hypothesis $h'' = 2$ (by the definition of inconsistency predicate *Inc*). Finally, the predicate *Evd* describes a relevant S-TLA⁺ system state (a valuation of some system variables) which is of capital importance especially in fulfilling forensic investigation objectives. Its use will be demonstrated afterwards in section 4.2.

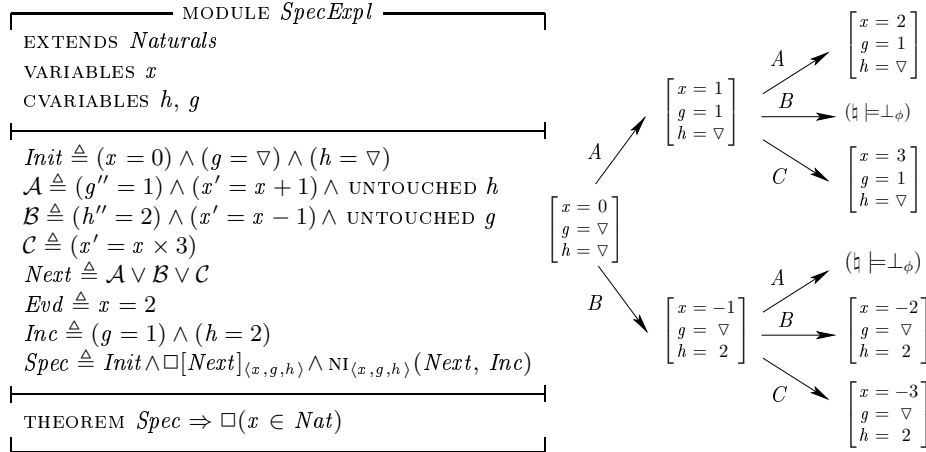


Fig. 3. Standard form of a S-TLA⁺ specification and a relative behavior fragment

The second part of Figure 3 describes a fragment from the set of possible system behaviors relative to formula *Spec*. For a successive execution of *A* followed by *B*, two successive hypotheses are generated: $g'' = 1$ followed by $h'' = 2$.

This is an unacceptable execution as it drives to a state where the S-TLA inconsistency predicate Inc will be true. Besides, a successive execution of action A followed by C is legitimate.

4.2 Computer Forensic Investigation Using S-TLA⁺

A scenario fragment component as modeled in Section 2 matches well the form of a S-TLA⁺ action. In fact, pre-conditions, generated hypotheses, and actions which represent the context of a scenario fragment can be described respectively by state-predicates, relations between assumed and non-assumed variables, and relations between primed and unprimed variables.

A digital forensic evidence can take the form of a temporal property (e.g., a hacked system is issuing every so often an outbound connection to send sniffed passwords), or an undesirable state of a system component (e.g., an altered file is a violation of the integrity property). These two forms can be specified in S-TLA⁺ using temporal formulas, and state predicates, respectively.

An expected hacking scenario is a disjunction of scenarios fragments (i.e., S-TLA⁺ actions) denoting possible hacking events starting from a state representing a safe system and ending in a state satisfying the digital evidence(s). The core S-TLA logic works by infinitely selecting the suitable scenario fragment that copes with the attained system behavior, such that no inconsistency is holding and composing it with the previous ones into potential hacking scenarios.

5 S-TLC: A Model Checker for S-TLA⁺ Specifications

To automate the proof in the context of forensic investigation, we propose S-TLC as an automated verification tool for S-TLA⁺ specifications with a stress on the handling of hypotheses and an improvement in the states space representation. S-TLC is somehow an extension to the Model Checker TLC[6], which checks S-TLA⁺ specifications for errors such as silliness, invariance properties violation, and deadlock [6, chapter 14]. In the following, we emphasize on the contributions and changes in S-TLC, namely state computation and scenario inference.

5.1 S-TLC's States Space Representation

Given two different states that represent respectively a valuation ($x = 1$) of the variable x under two possible sets of hypotheses ($h = 1 \wedge g = 2$) and ($h = 3 \wedge g = 3$). Representing a state as a valuation of all its variables (as in Figure 3) will involve a representation of two different states ((1, 1, 2) and (1, 3, 3)) in the generated scenarios. We propose a more developed and optimal representation involving two notions: *node core* and *node label*. The core of a node represents a valuation of the entire non-constrained variables, and the node label represents the potential sets of hypotheses (a set of hypotheses is a valuation of the entire constrained variables) under which the node core is reached. The node label is represented and maintained in a way akin to the one used in the

Assumption Truth Maintenance System (ATMS [10]). Precisely, a node label is a set of environments and an environment is a set of hypotheses. The previous example will thus involve only one node represented by $1\{(1, 2), (3, 3)\}$ where 1 is the node core, (1, 2) and (3, 3) are both environments, and $\{(1, 2), (3, 3)\}$ is the node label. In the following, given a state t , we use t_n to denote its corresponding node core, t_c to describe its resulting environment, and $Label(G, t)$ to refer to its label in graph G .

5.2 Inferring Scenarios with S-TLC

The S-TLC Model Checker is described by Algorithm 1. It employs three data structures \mathcal{G} , \mathcal{U}_F and \mathcal{U}_B . The first refers to the reachability directed graph under construction generated during forward chaining and backward chaining phase. The last two are FIFO queues, containing states whose successors have not being yet computed respectively during forward and backward chaining phases. The algorithm assumes that a configuration file is done as input, it includes statements denoting that *Init* is the initial state predicate, *Next* is the next state relation, *Invariant* is a state-predicate to be satisfied by each reachable state, and *Inc* is the predicate to be equal false for all states of the system behavior, it represents the the set of S-TLA inconsistencies. Moreover, the specification is supposed to be made finite-state. To that effect the configuration file is presumed to include statements stating that *Constraint* is a predicate that asserts bounds on the set of reachable states, and *EvidenceState* is a predicate characteristic of a terminal state representing forensic evidences.

To append a node to the graph under construction, the algorithm uses function $Append(\mathcal{G}, t, t \mapsto s)$ to add a node t to graph \mathcal{G} with a pointer to its predecessor state s . Besides, a state s is attached to a FIFO queue \mathcal{U} using the function $Append(\mathcal{U}, s)$ and detached using the function $Tail(\mathcal{U})$. Moreover, a node t is joined to an existing node s inside the graph \mathcal{G} using the function $Join(\mathcal{G}, t \mapsto s)$. S-TLC works in three phases:

Initialization Phase: \mathcal{G} , as well as \mathcal{U}_F and \mathcal{U}_B are created and initialized respectively to empty set \emptyset and empty sequence $\langle \rangle$. During this step, each state satisfying the initial system predicate is computed and then checked whether it satisfies predicate *Invariant*. In that case, it will be appended to \mathcal{G} after computing its label, and pointing it to the *null* state. If the state does not satisfy *EvidenceState*, it will be attached to the unseen queue \mathcal{U}_F , otherwise, it will be considered as a terminal state and appended to \mathcal{U}_B in order to be retrieved in backward chaining phase.

Forward Chaining Phase: During this phase, the algorithm starts with \mathcal{U}_F equal to the set of initial system states. Afterwards and until the queue becomes empty, state s (representing the tail of \mathcal{U}_F) is retrieved and its successor states are computed. From the latter, for every state (denoted by t) satisfying *Constraint*, if *Invariant* is not satisfied, an error is generated and the algorithm terminates, otherwise t is appended to \mathcal{G} as follows:

- If t_n does not exist in \mathcal{G} , it is appended as a new node with a label equal to t_c and a predecessor equal to s_n . Then, t is appended to \mathcal{U}_B if it satisfies *EvidenceState*, otherwise it is attached to \mathcal{U}_F .
- If there exists a node x in \mathcal{G} which is the same as t_n and whose label includes t_c , then a conclusion could be made stating that t has been added previously to \mathcal{G} . In that case, a pointer is simply added from x to s_n .
- If there exists a node x in \mathcal{G} that is the same as t_n , but whose label does not include t_c , then the node label is updated in the following manner:
 1. t_c is added to $Label(\mathcal{G}, x)$.
 2. Any environment from $Label(\mathcal{G}, x)$, which is a superset of some other environment in this label, is deleted to ensure hypotheses minimality. Formally, an environment E_1 is a superset of E_2 in the same environment iff: $E_1(x) = E_2(x) \vee E_2(x) = \nabla$, where $E(x)$ represents the x^{th} value in E . An environment (8, 1, 3) is for instance a superset of (∇ , 1, 3).
 3. If t_c is still contained in the label of state x (meaning that it was not deleted in step (2)) then node x is pointed to s_n and node t is appended to \mathcal{U}_B if it satisfies *EvidenceState*. Otherwise, it is attached to \mathcal{U}_F .

Every node label is provided with the following four properties: 1) Soundness: a node x holds each environment E_i ; 2) Consistency: None environment E_i in $Label(\mathcal{G}, x)$ is inconsistent, preventing *Inc* from holding; 3) Completeness: every environment E is a superset of some E_i ; and 4) Minimality: no environment E_i is a proper subset of any other.

Forward chaining may generate many slices of global attacks scenarios, a great majority of them are useless due to further occurrence of inconsistencies or because they do not lead to evidence generation. Nevertheless, this may generate additional source of evidences and show the propagation steps of the attack.

Backward Chaining Phase: This phase helps obtaining potential and additional scenarios that could be the root causes for the set of available evidences. This phase starts with \mathcal{U}_B holding the set of terminal states; the ones that satisfied *EvidenceState* in forward chaining phase. Afterwards, and until the queue becomes empty, the tail of \mathcal{U}_B , described by t , is retrieved and its predecessor states (the set of states s_i such that (s_i, t) satisfies action *Next*) which are not terminal states and satisfy *Invariant* (States that do not satisfy *Invariant* are discarded because this phase does not aim to check whether a specification is correct or not but simply to generate additional explanations) and *Constraint* are computed. Each computed s is appended to \mathcal{G} as follows:

- If s_n does not exist in \mathcal{G} , a new node (set to s_n) is appended to the graph with a label equal to s_c . Afterwards, a pointer is added from t_n to s_n and s is appended to \mathcal{U}_B .
- If there exists a node x in \mathcal{G} which is the same as s_n , and whose label includes s_c , then s was added previously to \mathcal{G} . In that case a pointer is simply added from t_n to s_n and s is appended to \mathcal{U}_B .
- If there exists a node x in \mathcal{G} which is the same as s_n , but whose label does not include s_c , then the node label of x is updated in the following manner:

1. The environment s_c is added to $Label(\mathcal{G}, x)$, the label of state x .
2. Any environment from $Label(\mathcal{G}, x)$ which is a superset of some other environments in this label is deleted to ensure hypotheses minimality.
3. If s_c is still contained in the label of x then t is pointed to the predecessor state x and s is appended to \mathcal{U}_B .

The outcome of the three phases is a graph \mathcal{G} of the potential scenarios that lead to the collected evidences. It embodies different initial system states apart from the ones described by the specification. In fact, in the context of forensic investigation, an attack scenario could start from a legitimate system state, as well as from a previous system incident or instability.

6 Case Study

To make concrete the use of S-TLA⁺ and S-TLC in digital forensic investigation, we propose this case study which is an investigation of a standalone (disconnected from network) system that is compromised, where an illegal privileged access is detected. The system ran initially with two users accounts: a root and an unprivileged user. A straightforward examination by experts shows that the system security log is altered. The latter no longer contains more than a single unexpected record showing that the system root has closed its session.

6.1 S-TLA⁺ Specification Description

The following set of S-TLA⁺ actions is specified to represent hacking scenarios fragments. For the sake of readability, we ignore the fragments that will not be part of the whole expected scenarios.

- *LogAsUsr*: Using the hypothesis stating that the user password is a well-known word, an intruder guesses the password and gains access to the system, raising its privilege *localpr* from 0 to 1. Moreover, the pair $\langle \text{“usr”}, \text{“logon”} \rangle$ is appended to the sequence *log* to log such event. Note that 0 means there is no granted access, while 1 lets a user execute any non administrative command. Finally, 2 refers to the root privilege.

$$\begin{aligned} \text{LogAsUsr} \triangleq & \ \wedge \text{userhas}' = \text{“weakpwd”} \wedge \text{localpr} = 0 \wedge \text{localpr}' = 1 \\ & \wedge \text{log}' = \text{Append}(\text{log}, \langle \text{“usr”}, \text{“logon”} \rangle) \end{aligned}$$

- *InstSoft*: A user who gained an unprivileged access can install its own software, particularly, a vulnerability exploit tool.

$$\text{InstSoft} \triangleq \text{localpr} = 1 \wedge \text{addsft} = \text{“”} \wedge \text{addsft}' = \text{“exploit”}$$

- *ExpLclVuln*: Hypothesizing that there is a vulnerability in one of the installed super-user commands that could grant a privileged access, if exploited, the current user exploits such vulnerability and rises its privilege from 1 to 2. The system kernel updates sequence *log* in order to log the event.

$$\begin{aligned} \text{ExpLclVuln} \triangleq & \ \wedge \text{roothas}' = \text{“vulnbin”} \wedge \text{localpr} = 1 \wedge \text{addsft} = \text{“exploit”} \\ & \wedge \text{localpr}' = 2 \wedge \text{log}' = \text{Append}(\text{log}, \langle \text{“root”}, \text{“logon”} \rangle) \end{aligned}$$

Algorithm 1. S-TLC algorithm

Comment: Initialization phase

$\mathcal{G} \leftarrow \emptyset, \mathcal{U}_F \leftarrow \langle \rangle, \mathcal{U}_B \leftarrow \langle \rangle$

$S \leftarrow \{s_i \mid s_i \models \text{Init}\}$

For each $s \in S$

$\left\{ \begin{array}{l} \text{if } s \not\models \text{Invariant} \text{ then error, break} \\ \text{do } \left\{ \begin{array}{l} \text{if } s \models \text{Constraint} \text{ then } \left\{ \begin{array}{l} \text{Append}(\mathcal{G}, s_n, s \mapsto \text{null}), \text{Label}(\mathcal{G}, s_n) \leftarrow s_c \\ \text{if } s \models \text{EvidenceState} \text{ then Append}(\mathcal{U}_B, s) \text{ else Append}(\mathcal{U}_F, s) \end{array} \right. \end{array} \right. \end{array} \right.$

Comment: Forward chaining phase

While $\mathcal{U}_F \neq \langle \rangle$

$\left\{ \begin{array}{l} s \leftarrow \text{tail}(\mathcal{U}_F) \\ T \leftarrow \{t_i \mid ((s, t_i) \text{ satisfies the S-TLA}^+ \text{ action Next}) \wedge t_i \models \text{Constraint}\} \\ \text{For each } t \in T \\ \text{do } \left\{ \begin{array}{l} \text{if } t \not\models \text{Invariant} \text{ then error, break} \\ \text{if } \nexists x \in \mathcal{G} / t_n = x \\ \text{then } \left\{ \begin{array}{l} \text{Append}(\mathcal{G}, t_n, t_n \mapsto s_n), \text{Label}(\mathcal{G}, t_n) \leftarrow t_c \\ \text{if } t \models \text{EvidenceState} \text{ then Append}(\mathcal{U}_B, t) \text{ else Append}(\mathcal{U}_F, t) \end{array} \right. \\ \text{if } (\exists x \in \mathcal{G} / t_n = x) \text{ and } t_c \subseteq \text{Label}(\mathcal{G}, x) \text{ then Join}(\mathcal{G}, x \mapsto s_n) \\ \text{do } \left\{ \begin{array}{l} \text{if } (\exists x \in \mathcal{G} / t_n = x) \text{ and } t_c \not\subseteq \text{Label}(\mathcal{G}, x) \\ \left\{ \begin{array}{l} \text{Label}(\mathcal{G}, x) \leftarrow \text{Label}(\mathcal{G}, x) \cup t_c \\ \text{Delete any superset of hypotheses from Label}(\mathcal{G}, x) \end{array} \right. \\ \text{then } \left\{ \begin{array}{l} \text{if } t_c \in \text{Label}(\mathcal{G}, x) \\ \text{then } \left\{ \begin{array}{l} \text{Join}(\mathcal{G}, t_n \mapsto s_n) \\ \text{if } t \models \text{EvidenceState} \text{ then Append}(\mathcal{U}_B, t) \text{ else Append}(\mathcal{U}_F, t) \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right.$

Comment: Backward chaining phase

While $\mathcal{U}_B \neq \langle \rangle$

$\left\{ \begin{array}{l} t \leftarrow \text{tail}(\mathcal{U}_B) \\ S \leftarrow \{s_i \mid ((s_i, t) \text{ satisfies the S-TLA}^+ \text{ action Next}) \wedge (s_i \models \text{Invariant}, \text{Constraint}) \wedge (s_i \not\models \text{EvidenceState})\} \\ \text{For each } s \in S \\ \text{do } \left\{ \begin{array}{l} \text{if } \nexists x \in \mathcal{G} / s_n = x \\ \text{then } \left\{ \begin{array}{l} \text{Append}(\mathcal{G}, s_n, t_n \mapsto s_n) \\ \text{Label}(\mathcal{G}, s_n) \leftarrow s_c \\ \text{Append}(\mathcal{U}_B, s) \end{array} \right. \\ \text{do } \left\{ \begin{array}{l} \text{if } (\exists x \in \mathcal{G} / s_n = x) \text{ and } s_c \subseteq \text{Label}(\mathcal{G}, x) \text{ then } \left\{ \begin{array}{l} \text{Join}(\mathcal{G}, x \mapsto s_n) \\ \text{Append}(\mathcal{U}_B, s) \end{array} \right. \\ \text{if } (\exists x \in \mathcal{G} / s_n = x) \text{ and } s_c \not\subseteq \text{Label}(\mathcal{G}, x) \\ \left\{ \begin{array}{l} \text{Label}(\mathcal{G}, x) \leftarrow \text{Label}(\mathcal{G}, x) \cup s_c \\ \text{Delete any superset of hypotheses from Label}(\mathcal{G}, x) \end{array} \right. \\ \text{then } \left\{ \begin{array}{l} \text{if } s_c \in \text{Label}(\mathcal{G}, x) \text{ then Join}(\mathcal{G}, t_n \mapsto s_n), \text{Append}(\mathcal{U}_B, s) \end{array} \right. \end{array} \right. \end{array} \right.$

- *OffBrForce*: Hypothesizing that the algorithm used to hash the account's passwords is weak, a user reads the file containing the password hashes and brute-forces the root password off-line (outside the current system). It succeeds thus in escalating its privilege.

$$\text{OffBrforce} \triangleq \wedge \text{roothas}'' = \text{"pwdhashcomp"} \wedge \text{localpr} = 1 \wedge \text{localpr}' = 2 \\ \wedge \text{log}' = \text{Append}(\text{log}, \langle \text{"root"}, \text{"logon"} \rangle)$$

- *ChangeID*: Hypothesizing that the root password is equal to the user's, the user changes its identity to the root by providing the correct password. Consequently, its privilege rises from 1 to 2, and the event is logged.

$$\begin{aligned} \text{ChangeID} \triangleq & \wedge \text{roothas}' = \text{"pwdequser"} \wedge \text{localpr} = 1 \wedge \text{localpr}' = 2 \\ & \wedge \text{log}' = \text{Append}(\text{log}, \langle \text{"root"}, \text{"logon"} \rangle) \end{aligned}$$

- *ExtSoft*: Given a software installed on the system for security auditing purpose, the user copies one binary command from those that come with it to be used maliciously as an exploit tool.

$$\text{ExtSoft} \triangleq \text{localpr} = 1 \wedge \text{addsft} = \text{"audittool"} \wedge \text{addsft}' = \text{"exploit"}$$

- *CleanLog*: A privileged user can clean the log file content.

$$\text{CleanLog} \triangleq \text{localpr} = 2 \wedge \text{log} \neq \langle \rangle \wedge \text{log}' = \langle \rangle$$

- *DelSoft*: A privileged user can delete the whole tools unexpectedly installed.

$$\text{delSoft} \triangleq \text{localpr} = 2 \wedge \text{addsft} \neq \text{""} \wedge \text{addsft}' = \text{""}$$

- *Exit*: The user logs off, its privilege goes down to 0 and the event is logged.

$$\text{Exit} \triangleq \text{localpr} = \wedge \text{localpr}' = 0 \wedge \text{log}' = \text{Append}(\text{log}, \langle \text{"root"}, \text{"logoff"} \rangle)$$

Inconsistency defined as: $\text{userhas} = \text{"weakpwd"} \wedge \text{roothas} = \text{"pwdequser"}$, states that a system state should not be reached under a conjunct of the following two hypotheses: a) the user password is a well-known word and b) the root password is equal to the user one. In fact, the forensic investigator is sure that the root password fulfills a strong password policy. The available evidence is described by predicate $\text{EvidenceState} \triangleq \text{Head}(\text{log}) = \langle \text{"root"}, \text{"logoff"} \rangle$, which states that the finite sequence log encloses only one record equal to $\langle \text{"root"}, \text{"logoff"} \rangle$.

The system under investigation is specified by a S-TLA⁺ formula Spec similarly to the form described in section 4.1, where Init describes the initial system state (empty log file, no unexpected tool installed, no granted access).

$$\text{Init} \triangleq \text{localpr} = 0 \wedge \text{log} = \langle \rangle \wedge \text{addsft} = \text{""} \wedge \text{userhas} = \nabla \wedge \text{roothas} = \nabla$$

6.2 Investigation Using S-TLC

Figure 4 describes the results generated by S-TLC until the forward chaining phase. It outlines two different system states (the ones which are encircled) satisfying predicate EvidenceState , where one of them shows a new generated evidence as an exploit tool installed by the malicious user to exploit a local vulnerability. These two evidences can be generated under two possible set of hypotheses: 1) the user password is weak and one of the installed system commands contains a vulnerability that grants a privileged access; and 2) the user password and the password hashing algorithm are both weak. Two main possible scenarios may be distinguished in this phase:

1. An intruder guesses the weak user password and gains an unprivileged access. Afterwards, it exploits a weakness in the password hashing algorithm and

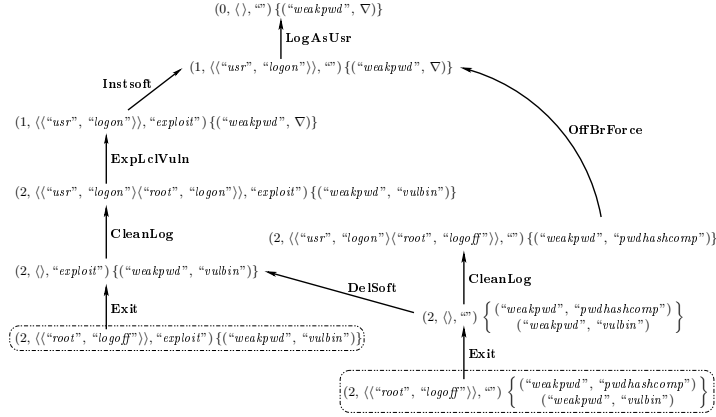


Fig. 4. Scenarios generated in forward chaining phase

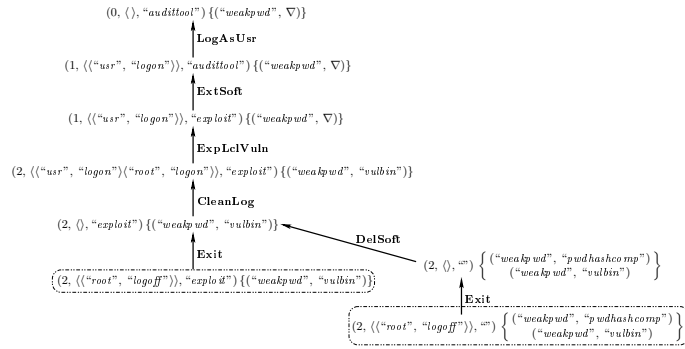


Fig. 5. Scenarios generated in backward chaining phase

succeeds in escalating its privilege by performing an offline brute-force of the root password. It cleans its logged activity and logs off from the system. Fortunately, the latter activity is logged.

2. An intruder guesses a weak user password and logs in to the system, gaining an unprivileged access. After that, it installs a malicious tool and exploits a vulnerability in one of the installed super-user commands, obtaining thus a privileged access. Before cleaning the log file and leaving the system, the intruder either deletes its installed tool or leaves such kind of evidence.

The generated scenario prevents inconsistency from occurring. In fact, action *ChangeID* does not belong to the scenario since it contains a hypothesis that is inconsistent with the one occurring in *LogAsUsr* according to the definition of predicate *Inconsistency*. The graph of Figure 5 is the graph generated after the execution of forward and backward chaining phases. For readability reasons, it

shows only the new generated scenarios compared to the ones of Figure 4. Mainly, a new scenario is added. It strongly resembles to the second one generated in forward chaining phase, except that the system is initially housing a security auditing software and the hacker is using one of the commands that come with such software as an exploit tool, instead of installing its own one.

7 Conclusion

We proposed in this paper a novel formal logic-based language entitled S-TLA⁺ to achieve a tremendous aspect in digital forensic investigation: the reconstruction of potential hacking scenarios and the providing of new evidences that could complement the available ones. S-TLA⁺ uses a formalism that allows handling hypotheses whenever there is a lack of details to demonstrate some part of an attack scenario. We have also described S-TLC as a new automated formal verification tool that is able to handle S-TLA⁺ specifications. Its main advantage lies in its robustness in managing hypotheses and representing states. Considering implementing and testing this tool represents a continuation of this work.

References

1. Kruse, W.G., Heiser, J.G.: Computer Forensics: Incident Response Essentials. Pearson Education (2001)
2. Stephenson, P.: Modeling of post-incident root cause analysis. *International Journal of Digital Evidence*, Vol. 2, No. 2 (2003)
3. Elsaesser, C., Tanner, M.C.: Automated diagnosis for computer forensics. tech. rep., The MITRE Corporation (2001)
4. Stallard, T., Levitt, K.: Automated analysis for digital forensic science: Semantic integrity checking. *Proceedings of the 19th Annual Computer Security Applications Conference* (2003)
5. Rekhis, S., Boudriga, N.: A Formal Logic-based Language and an Automated Verification Tool For Computer Forensic Investigation. *Proceedings of the 20th ACM Symposium on Applied Computing (SAC 2005)* (2005)
6. Lamport, L.: *Specifying Systems: The TLA⁺ Language and Tools for Hardware and Software Engineers*. Addison-Wesley (2002)
7. Yu, Y., Manolios, P., Lamport, L.: Model checking TLA⁺ specifications. *Conference on Correct Hardware Design and Verification Methods* (1999) 54–66
8. Keppens, J., Zeleznikow, J.: A model based reasoning approach for generating plausible crime scenarios from evidence. *Proceedings of the 9th International Conference on Artificial Intelligence and Law* (2003)
9. Lamport, L.: The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, Vol. 16 (1994) 872–923
10. de Kleer, J.: An assumption-based TMS. *Artificial Intelligence*, Vol. 28, No. 2 (1986) 127–162