A Formal Logic-based Language and an Automated Verification Tool For Computer Forensic Investigation

Slim REKHIS
Communication Networks and Security Research Lab (SUP’COM)
University of Carthage November 7th, Tunisia
smr@certification.tn

Noureddine BOUDRIGA
Communication Networks and Security Research Lab (SUP’COM)
University of Carthage November 7th, Tunisia
nab@supcom.rnu.tn

ABSTRACT
In this paper, a formal logic-based language, called S-TLA⁺, is proposed for computer forensic investigation. It allows an unambiguous description of evidences, a modeling of the forensic experts' knowledge in the form of hacking scenarios fragments, and a reasoning capability with uncertainty by filling in potential lack of data with hypotheses. The proposal is complemented by an automated formal verification tool, called S-TLC which helps exploring additional evidences and checks whether there are plausible hacking scenarios that meet the available evidences.

Keywords
Formal Forensic Investigation, Temporal Logic of Security Actions, S-TLA⁺, S-TLC.

1. INTRODUCTION
After the occurrence of digital security incident, computer forensic investigators start their analysis focusing on collecting evidences, determining the techniques and scenarios used to cause the incident, and building proofs from the collected information to enable hackers prosecution. Achieving these objectives with a cursory observation on the compromised system is a challenging problem. Computer investigation efforts may depend upon a number of factors including: 1) how good the system under investigation has been initially prepared to handle forensic analysis; 2) the investigators' skills and their familiarity with attack techniques; and 3) the complexity of the conducted attacks. To overcome the above constraints, computer forensic investigation may require both formalization and automation for accuracy and practicality. This is time-saving and allows an unambiguous representation of forensic investigators' knowledge and observations. It also makes the generated investigation deductions relevant, even with a huge amount of data.

Formal reasoning on digital forensic investigation has interested a few works that differ on the techniques and methodologies they used. [2] 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 the attack activity is logged, which is in contradiction with the fact that attack scenarios may subvert logging daemon and alter logs before hackers leave the system. [7] 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. Moreover, while the system enables automation, it does not allow disclosing the performed hacking scenario. [6] proposed an approach to automatically reconstruct a forensic evidence from a set of scattered fragments using a context modeling technique, where probabilities related to fragment adjacency are computed and pruning heuristics are used to compute the optimal ordering. One drawback to this approach is its dependency on the evidence type.

We propose in this work a new logic-based language along with an automated verification tool for formal computer forensic investigation of digital security incidents. The work is based on the enhancement of the formal specification language TLA⁺ and its model checker TLC [5]. TLA⁺ has shown a good track record in describing concurrent and asynchronous systems. Our new language, called S-TLA⁺, is proposed to enable a formal and unambiguous description of the available evidences, a modeling of the forensic experts' knowledge in the form of hacking scenarios fragments, and a reasoning technique on compromised systems with uncertainty by adding forward hypotheses to fill in any potential lack of details. S-TLC, as a new designed automated formal verification tool, helps exploring additional evidences and checking whether there are hacking scenarios that meet the available evidences.

Our contribution is 3-fold. First, S-TLA⁺ takes advantage from the richness of the formal specification language TLA⁺ to support description of scenarios and evidences using temporal modalities. Second, S-TLC keeps the core components of TLC and represents a promising tool able to handle a generic class of S-TLA⁺ specifications. Third, such tool provides a computer-assisted diagnosis that reduces the computer investigator efforts and produces a for-
mal proof regarding malicious hacking activities without a need for advanced skills. The remaining part is organized as follows: First, the temporal logic of actions is reviewed and the novel S-TLA logic along with the formal specification language S-TLA\textsuperscript{+} is defined. Section 3 describes the automated verification tool S-TLC, and demonstrates how hacking scenarios are inferred both using forward and backward chaining. In Section 4, the proposal is exemplified by a case study. Finally, the work is concluded in Section 5.

2. TEMPORAL LOGIC OF SECURITY ACTIONS

We describe here the Temporal Logic of Actions and its supported language TLA\textsuperscript{+} [5], as they serve as a starting point. Then we discuss the Temporal Logic of Security Actions.

2.1 Reviewing TLA/TLA\textsuperscript{+}

TLA is a state-based logic that was introduced by L. Lamport for the specification of distributed and asynchronous systems, [4]. It allows the description of states and transitions. A state is a mapping from the set of all variable names to the collection of all possible values. A TLA formula is built from state function using Boolean operators in addition to three new operators (\textsuperscript{′}, \textsuperscript{\prime}, and \textsuperscript{□}). State function and state predicate are of the same syntax. The formula assigns a value to each state, the latter is a boolean expression that is true or false for a state. A transition is described by an action that is a relation between an old state, and a new state. Semantically, it is true or false for a pair of states \((s, t)\); and syntactically, it represents a boolean expression containing primed and unprimed variables (primed refer to the variables of the new state and \textsuperscript{′}the prime operator). In TLA, a system is specified by a formula of the form \(3x : \text{Init} \land \Box[\forall v \land L]\), describing the set of all its authorized behaviors. It describes a system whose initial behavior satisfies \text{Init} and where every state satisfies the next state relation \(\Box\), or leaves the tuple of specification variables unchanged. The infinite behavior of the system is constrained by the liveness property \(L\) (written as a conjunction of weak and strong fairness conditions on actions).

TLA\textsuperscript{+} is a high-level language that is based on TLA and contains notations of set theory and syntactic structuring mechanisms [5]. TLA\textsuperscript{+} provides a precise syntax and module system for writing specifications. A module represents the lowest granular part of a TLA\textsuperscript{+} specification, it contains three main sections embedding declarations, definitions, and assumptions.

2.2 Enhancing TLA

Suppose a formal system description that should involve a detail (value progress) of its \(n\) dependent variables, while currently only the behavior of \(m\) system variables (\(m < n\)) is understood. Such lack of details happens often during computer forensic investigation. To overcome such limitation, it is conceivable to use a formalism that supports a reasoning with uncertainty, to let enunciating hypotheses whenever there is missing information. TLA is unable to support such kind of reasoning. To this end, we define S-TLA and its supported S-TLA\textsuperscript{+} language that is suitable for formal modeling of digital postmortems. In a thoughtfully manner, we make TLA able to describe a system progress from a state to another, further to the execution of an action \(A\) under hypothesis \(H_A\).

S-TLA constrained variables: Distinguishing variables whose value is set according to a hypothesis, from the remaining specification variables is of great importance. In fact, we aim to handle them differently, as we are looking to reach a given system state under a minimal set of hypotheses. This will be significantly realized during verification. Moreover a variable whose value is set according to a hypothesis could not be valued differently afterwards all through the system behavior. Such type of variable is a bit different from the two TLA flexible and rigid variables [5]. Thus, we introduce a new set of variables called constrained variables set \(V_C\) to encompass the variables representing hypotheses. This set of variables should be disjoint from sets \(V_F\) and \(V_R\) that represent flexible and rigid variables sets, respectively.

S-TLA state: The semantic of the new S-TLA state remains always a valuation of variables, it is an assignment from the collection \(\text{Val}\) of values to the set \(Var = V_F \cup V_R \cup V_C\) of variables including the new constrained variables set \(V_C\). A state can thus give information on the set of enunciated hypotheses that let it being reachable from the initial system state.

S-TLA assumption operator `\textsuperscript{□}`: Every time the TLA prime operator `\textsuperscript{′}` is applied to a flexible variable, it changes its value independently of the previous values that have been effected to it in the previous system states. Such operator cannot thus be applied to a constrained variable as defined above. We introduce a new S-TLA operator entitled assumption operator `\textsuperscript{□}` to denote the value of a constrained variable in the new state. 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.

The S-TLA fictive value `\textsuperscript{□}`: While a state is a valuation of the wholeness variables, a new fictive TLA value described by the symbol `\textsuperscript{□}` 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 value that is different from `\textsuperscript{□}` means that there is an enunciated hypothesis to reach the related state.

S-TLA inconsistency: We formally define an S-TLA inconsistency `\textsuperscript{⊥}` as a predicate containing constrained variables, constants, and constants operators to denotes an unwanted condition (informally, it reflects a combination of hypotheses). Semantically, it is true or false for a state. If an inconsistency is true for a state \(t\), then the system transition on the way to that state should not be followed.

S-TLA action and hypothesis: An S-TLA action is a conjunct of two expressions. The first expression is facilitative, of type boolean, denotes a hypothesis, and contains assumed and non-assumed variables. The second is an old TLA action containing primed and unprimed variables. In the case where an S-TLA action \(A\) looks like a TLA action (no hypothesis is enunciated), its semantic remains the same as defined in [5]. Semantically, given an inconsistency denoted by `\textsuperscript{⊥}`, an S-TLA action \(A\) is true for a pair of states \(\langle s, t \rangle\) if, and only if \((A(s(x)/x))\) expression means \(x\) is replaced by \(s(x)\), the value of \(x\) in state \(s\), in action \(A\):

\[-A(\forall v \in V_F : s(v)/v, t(v)/v') = true\]
S-TLA specification formula: To ensure that $\perp_\phi$ is false in all system behaviors whenever an action $\mathcal{N}$ is executed, we introduce the “No Inconsistency” definition as

$$\text{NL}(\mathcal{N}, \perp_\phi) \triangleq \text{enabled } \mathcal{N} \implies \langle \mathcal{N} \rangle_{\phi} \wedge \neg \perp_\phi$$

The latter will be part the standard S-TLA specification formula $\phi$ defined as follows:

$$\phi \triangleq \exists x : \text{Init} \wedge \Box[\mathcal{N}]_{\phi} \wedge L \wedge \text{NL}(\mathcal{N}, \perp_\phi)$$

Except the quoted syntactically and semantically modifications, the remaining TLA notions including Fairness, stuttering, and temporal modalities are preserved.

2.3 Writing S-TLA$^+$ specifications

We define S-TLA$^+$ as a language for writing specifications in S-TLA, it embodies TLA$^+$ with some add-ons in the module structure and in the constant and non constant operators. To describe S-TLA$^+$, we concentrate only on the introduced modifications as outlined hereinafter:

Module-Level constructs: \texttt{cvariables cvar}_1, \ldots, \texttt{cvar}_n

Adds to the current module the declaration of constrained variables.

Non constant S-TLA$^+$ operators:

Given a constrained variable $h$:

- $h''$ [Denotes the value of $h$ in the next state]
- $\text{untouched } h$ [Replace the expression $h'' = h$]

Constant S-TLA$^+$ operators:

- $\nabla$ [A fictive value that represents the initial constrained variable value, before a hypothesis is enunciated]

$$\text{MODULE SpecExp}$$

$$\text{EXTENDS Naturals}$$

$$\text{VARIABLES } x$$

$$\text{cvariables } h, g$$

$$\text{Init} \triangleq (x = 0) \wedge (g = \nabla) \wedge (h = \nabla)$$

$$\text{A} \triangleq (g'' = 1) \wedge (x' = x + 1) \wedge \text{untouched } h$$

$$\text{B} \triangleq (h'' = 2) \wedge (x' = x - 1) \wedge \text{untouched } g$$

$$\text{C} \triangleq (x' = x \times 3)$$

$$\text{Next} \triangleq \text{A} \lor \text{B} \lor \text{C}$$

$$\text{End} \triangleq x = 2$$

$$\text{Inc} \triangleq (g = 1) \wedge (h = 2)$$

$$\text{Spec} \triangleq \text{Init} \land \Box[\text{Next}]_{(s, q, h)} \land \text{NL}((s, q, h), (\text{Next}, \text{Inc}))$$

$$\text{THEOREM } \text{Spec} \Rightarrow (x \in \text{Nat})$$

Figure 1: Standard form of a S-TLA$^+$ specification

Figure 1 illustrates a typical S-TLA$^+$ specification module, \texttt{SpecExp}. In the initial state there is no enunciated hypotheses and the constrained variables $g$ and $h$ are both equal to $\nabla$. 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 $\text{Inc}$). Finally, the predicate $\text{End}$ describes a relevant S-TLA$^+$ system state (a valuation of some system variables) which is of capital importance especially in specifying forensic evidences as it will be demonstrated afterwards in section 4.

3. AUTOMATED VERIFICATION USING S-TLC

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. S-TLC is somehow an extension to TLC [5, chapter 14], the Model Checker for TLA$^+$. S-TLC involves two notions: \texttt{node core} and \texttt{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 (i.e., 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 [1]). Precisely, a node label is a set of environments, which are sets of hypotheses. The graph generated by S-TLC is built by ensuring that a given node is reached under minimal sets of hypotheses, while a straightforward reading of the node label indicates the alternatives (hypotheses sets) under which a node is reachable.

3.1 Inferring scenarios with S-TLC

Given a state $t$, we use $t_n$ to denote its corresponding node core, $t_c$ to describe its resulting environment, and \texttt{Label}(G, t) to refer to its label in graph $G$. The S-TLC algorithm employs three data structures $\mathcal{G}$, $\mathcal{U}_F$ and $\mathcal{U}_B$; $\mathcal{G}$ refers to the reachability directed graph under construction, $\mathcal{U}_F$ and $\mathcal{U}_B$ are FIFO queues containing states whose successors are not yet computed, respectively during forward and backward chaining phases. The S-TLC Model Checker 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 predicate $\text{Init}$ is computed and then checked whether it satisfies the invariant predicate $\text{Invariant}$. On satisfaction, it is appended to $\mathcal{G}$ with a pointer to the null state and a label equal to the set of hypotheses relative to the current state. On failure, an error is generated. If the state does not satisfy the evidence predicate $\text{EvidenceState}$, it is attached to the unseen queue $\mathcal{U}_F$, otherwise it is considered as a terminal state and appended to $\mathcal{U}_B$ to be retrieved in backward chaining.

- **Forward chaining phase**: All the scenarios that originate from the set of initial system states are inferred in forward chaining. This involves the generation of new sets of hypotheses and evidences that are consequent to these scenarios. During this phase and until the queue becomes empty, state $s$ is retrieved from the tail of $\mathcal{U}_F$ and its successor states are computed. For every successor state $t$ satisfying the predicate (specified to assert bounds on the set of reachable states) $\text{Constraint}$, if the predicate $\text{Invariant}$ is not satisfied, an error is generated and the algorithm termi-
nates, otherwise state \( t \) is appended to \( \mathcal{G} \) as follows:

1. If \( t_0 \) does not exist in \( \mathcal{G} \), a new node (set to \( t_n \)) is appended to the graph with a label equal to \( t_0 \) and a predecessor equal to \( s_n \). State \( t \) is appended to \( \mathcal{U}_B \) if satisfies predicate EvidenceState, otherwise it is attached to \( \mathcal{U}_P \).

2. If there exists a node \( x \) in \( \mathcal{G} \) that is equal to \( t_n \) and whose label includes \( t_n \), then a conclusion could be made stating that node \( t \) was added previously to \( \mathcal{G} \). In that case, a pointer is simply added from \( x \) to the predecessor state \( s_n \).

3. If there exists a node \( x \) in \( \mathcal{G} \) that is equal to \( t_n \), but whose label does not include \( t_n \), then the node label is updated as follows: First, \( t_n \) is added to \( \text{Label}(\mathcal{G}, x) \). Second, any environment from \( \text{Label}(\mathcal{G}, x) \), which is a superset of some other elements in this label, is deleted to ensure hypotheses minimality. Third, if \( t_n \) is still in \( \text{Label}(\mathcal{G}, t) \) then \( x \) is pointed to the predecessor state \( s_n \) and node \( t \) is appended to \( \mathcal{U}_B \) if satisfies predicate EvidenceState. Otherwise, it is attached to \( \mathcal{U}_P \).

The resulting graph is a set of scenarios that end in any state satisfying the predicate EvidenceState and/or Constraint.

**Backward chaining phase:** All the scenarios that could produce states satisfying predicate EvidenceState, generated in forward chaining, are constructed. During this phase and until the queue becomes empty, the tail of \( \mathcal{U}_B \), described by state \( t \), is retrieved and its predecessor states (ie, the set of states \( s \), such that \( (s, t) \) satisfy action Next) which are not terminal states and satisfy the predicate Invariant (States that don’t satisfy predicate Invariant are discarded because this step aims simply to generate additional explanations) and Constraint are computed. Each computed state \( s \) is appended to \( \mathcal{G} \) as follows:

1. If \( s_n \) is not in \( \mathcal{G} \), a new node (set to \( s_n \)) is appended to \( \mathcal{G} \) with a label equal to the environment \( s_n \). Then, a pointer is added from node \( t_n \) to \( s_n \) and state \( s \) is appended to \( \mathcal{U}_B \).

2. If there is \( x \) in \( \mathcal{G} \) that is equal to \( s_n \), and whose label includes \( s_n \), then it is stated that node \( s \) was been added previously to \( \mathcal{G} \). In that case a pointer is simply added from \( t_n \) to the predecessor state \( s_n \) and \( s \) is appended to \( \mathcal{U}_B \).

3. If there is \( x \) in \( \mathcal{G} \) that is equal to \( s_n \), but whose label doesn’t include \( s_n \), then \( \text{Label}(\mathcal{G}, x) \) is updated as follows: First, \( s_n \) is added to \( \text{Label}(\mathcal{G}, x) \). Second, any environment from \( \text{Label}(\mathcal{G}, x) \) which is a superset of some other elements in this label is deleted to ensure hypotheses minimality. Third, if \( s_n \) is still contained in the label of state \( x \) then the node \( t \) is pointed to the predecessor state \( s_n \) and the node \( s \) is appended to \( \mathcal{U}_B \).

The outcome of the three phases is a graph \( \mathcal{G} \) containing the set of possible causes relative to the collected evidences. It embodies different initial system states apart from those described by the specification.

### 4. CASE STUDY

We propose the following case study which is an investigation on a public Web server that was found stopped providing its service. To deal with complex, unguessable, and attack scenarios, where hackers may have altered any trace, we follow the approach used by [3]. We consider a hacking scenario as a combination of more generic and reusable fragments, which are basically described in advance without any a priori knowledge about the whole hacking scenario that is looked for. This contributes to the creation of scenarios progressively and allows security experts to interfere effectively. Using S-TLA\(^+\), we model every scenario fragment 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. For the sake of space, we will only describe some parts of the S-TLA\(^+\) specification, deemed important.

#### 4.1 Specification using S-TLA\(^+\)

The available evidence is described with predicate Evd. It uses function running to state that there is no running Web service on the TCP port srvport described as a specification parameter. Evd \( \triangleq \text{running}[\text{srvport}] \neq \text{"web"} \)

Besides, S-TLA\(^+\) actions are used to specify hacking scenario fragments. For readability reasons, we ignore those that are not part of the whole expected scenarios.

1. \( \text{RmtUnprExploit} \): If there is a web vulnerability that directly grants an unprivileged system access and provided that there is a running web service, the user privilege \( pr \) rises from 0 to 1 as the vulnerability is exploited. Note that 0 means a restricted user privilege, while 1 lets a user execute any command that the service is able to execute. Finally, 2 refers to the system administrator privilege.

\[ \text{RmtUnprExploit} \triangleq \ \land \ \text{webvul} \Rightarrow \text{"rmtunpr"} \]
\[ \land \exists x \in \text{Ports} : \text{running}[\text{srvport}] = \text{"web"} \]
\[ \land pr = 0 \land pr' = 1 \]

2. \( \text{RmtPrExploit} \): This resembles to the previous fragment, except that it assumes the existence of a web vulnerability that directly grants a privileged access which leads the user privilege to raise directly to the administrator’s one as the vulnerability is exploited.

\[ \text{RmtPrExploit} \triangleq \ \land \ \text{webvul} \Rightarrow \text{"rmtpr"} \]
\[ \land \exists x \in \text{Ports} : \text{running}[\text{srvport}] = \text{"web"} \]
\[ \land pr = 0 \land pr' = 2 \]

3. \( \text{EscaladePr1} \): Hypothesizing that there is a local configuration error in how local users are affected to groups. The user privilege rises from 1 to 2 as the local configuration error is exploited.

\[ \text{EscaladePr1} \triangleq \land \text{cferrorin} = \text{"usrgrpconf"} \]
\[ \land pr = 1 \land pr' = 2 \]

4. \( \text{EscaladePr2} \): Similar to the previous fragment, except that the hypothesis concerning configuration error affects the running system update utility.

\[ \text{EscaladePr2} \triangleq \land \text{cferrorin} = \text{"systemupdateutility"} \]
\[ \land pr = 1 \land pr' = 2 \]

5. \( \text{InstallBackdoor} \): Given no installed backdoor yet and a provided administrator privilege, a unused port is chosen at random and a shell is bound on it.

\[ \text{InstBackdoor} \triangleq \land \text{running}[\text{x}] = \text{"shell"} \]
\[ \land pr' = 2 \]
\[ \land \exists x \in \text{Ports} : \text{running} \notin \{\text{srmv} \} \]

6. \( \text{RemoveTelnetAuthentication} \): Telnet service is recon-
figured to disable user authentication. The result of this operation is a shell bound to the same port. The system behaves thus like a backdoored system. Such action could be performed unless the current user is provided with an administrator privilege.

\[ \text{RmTelnetAuth} \equiv \land pr = 2 \land \exists x \in \text{Ports} : \land \text{running}[x] = "\text{telnet}" \land \text{running}' = [\text{running EXCEPT } ![x] = "\text{shell}"] \]

7) DoSWeb: Whenever the running user is equipped with an administrative privilege, it could stop the web service, creating thus a denial of service state.

\[ \text{DoSWeb} \equiv \land \text{running}[\text{srvrport}] \neq "\text{null}" \land pr = 2 \land \text{running}' = [\text{running EXCEPT } ![\text{srvrport}] = "\text{null}"] \]

An S-TLA\(^+\) inconsistency is specified by predicate Inc to state that a system state should not be reached under a conjunct of two types of hypotheses: a) there is a web vulnerability and b) the system update utility is running. In fact, whenever a system update utility is running, it is unfeasible to have any persistent fixed vulnerability.

\[ \text{Inc} \equiv \land \text{cferrormi} = "\text{systemupdateutility}" \land \lor \text{webvul} = "\text{rmtpr}" \lor \text{webvul} = "\text{rmtunpr}" \]

The system under investigation is specified by a S-TLA\(^+\) formula (c.f. section 2.2) where the initial system state is described by the predicate Init denoting a secure server providing only a web service, and where every permissible action is one among the seven specified scenario fragments, such that no inconsistency (as defined above) holds.

### 4.2 Automated investigation using S-TLC

The execution of the described S-TLC algorithm with parameters PortNumber and srvrport respectively equal to 2 and 1, until forward chaining (c.f. Figure 2) brings out two different system states (the ones which are encircled) representing forensic evidences. The second state shows a new additional evidence. It denotes an installed backdoor on another TCP port (precisely port number 2), which provides a privileged shell on access. These evidences are generated under two possible set of hypotheses: 1) there is a remote unprivileged vulnerability in the installed web service and a local configuration error in creating user groups and 2) there is a remote privileged vulnerability in the installed web service.

Inconsistency is prevented from occurring, in fact action EscaladePr2 doesn’t belong to the scenario since it contains a hypothesis that is inconsistent with the one occurring in RntUnprExploit.

During backward chaining a new scenario is appended. It states that the system is initially running telnet and web services. A hacker tries to get administrative privilege in the same way as in the previous scenarios. It introduces misleading system modifications in the configuration of telnet service by removing the authentication phase, letting the system grant a shell to any connected user and thus behaves like a backdoored system.

### 5. CONCLUSION

We have proposed a novel formal logic-based language and an automated verification tool, called S-TLA\(^+\) and S-TLC. S-TLA\(^+\) uses a robust formalism that allows advancing hypotheses whenever there is some lack of details to demonstrate some part of an attack scenario. S-TLC presents efficient robustness in managing hypotheses and representing states. Two main drawbacks of such approach could be perceived. First, the use of labels with model checking could become painful as they may grow wider requiring thus a great amount of memory space to represent individual states. Secondly, the use of backward chaining might affect considerably the system performance as this requires an additional resources in interpreting the set of S-TLA\(^+\) actions backwards. Nonetheless, it can be argued that the benefits of backward chaining are important in the sense that valuable additional explanations could be generated.

### 6. REFERENCES