



CNAS REPORT

Printed January 2008

CNAS-2008-004
Public

Supersedes CNAS-2007-004
Dated Dec. 2007

Execution-based Digital Investigation on Compromised Systems with Automated Hypotheses Generation

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.

Execution-based Digital Investigation on Compromised Systems with Automated Hypotheses Generation

Slim REKHIS and Nouredine BOUDRIGA

CN&S Research Lab., University of the 7th of November at Carthage, Tunisia
slim.rekhis@gmail.com, nab@supcom.rnu.tn

Abstract— This paper proposes an execution-based formal approach for digital forensic investigation. It considers an attack scenario as a sequence of legitimate and malicious actions. Using a library of potential hypotheses, a library of legitimate actions and a formal description of the system under investigation, our approach works by rebuilding the attack scenarios in forward and backward chaining manner. During reconstruction, malicious events are generated based on selected hypotheses. The execution graph is produced with an enhancement in states representation and hypotheses management. A case study on a compromised FTP server is provided to show how our method performs practically.

I. INTRODUCTION

Facing the increase and sophistication of security attacks, security experts have shown a significant interest in an emerging field of networked computers security, the digital forensic investigation of security incidents [2]. Such discipline aims at meeting a set of objectives including: evidence collection to carry out digital postmortems, rebuilding the potential conducted attack scenarios, and proving hacker's malice starting from the collected evidences. The complexity of attacks, however, makes investigation a challenging issue, as attackers try to remove, hide, or alter any sign of their action.

Digital forensic investigation can be aided by formal specifications and proof automation. Specification allows explicit and unambiguous representation of the investigator's knowledge and observations. Proof automation makes the investigation deductions relevant even in the presence of a huge amount of data. Formal reasoning in digital forensic investigation has interested a few research works. [1] presented an automated diagnosis system that identifies attacks 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 daemons and alter logs before hackers leave the system. [6] used an expert system with a decision tree to search through evidences for potential violations of invariant relationships between digital objects. The methodology roughly depends on the system time granularity and the degree of precision that the system uses to record time on objects. Moreover, while the system enables automation, it does not allow disclosing the performed hacking scenario.

In previous works [4] [5], we proposed a Temporal Logic of Security Actions (called S-TLA) to enable reasoning with uncertainty based on the use of hypotheses and allow unambiguous description of the set of available evidences and the elementary scenarios fragments that are compatible with the investigated system. To automate the potential attack scenarios reconstruction and the hypotheses management, we provided the Model Checker S-TLC. The approach presents three drawbacks: a) the attack scenario reconstruction may fail if an attack scenario fragment is missing within the library; and b) the technique requires that hypotheses be specified in conjunction with these fragments. Consequently, the choice of a hypothesis is imposed by the selection of the attack scenario fragment.

The paper proposes an execution-based formal approach for digital forensic investigation. The approach considers an attack scenario as a deviation from the normal behavior (known in advance by the investigator) and allows to rebuild the states history and the occurring events that led the system to a compromised state. Since describing all potential malicious events in advance is impractical, hypotheses need to be put forward in order to fill in this gap. Using a library of potential hypotheses and a formal description of the system under investigation, our approach works by rebuilding the attack scenarios in a forward and backward chaining processing. During reconstruction, hypotheses are selected from the library and malicious events are generated and combined to the legitimate actions to produce a proof regarding the performed attack. The execution graph is produced so that a straightforward reading of the node label gives an idea of the sets of hypotheses under which the state is reachable. Our contribution puts the ground work for a promising computer-assisted diagnosis tool that reduces the computer investigator's efforts and ambiguities.

The remaining of this paper is organized as follows: Section II defines the attack scenario model and highlights the role of libraries for digital investigation. Section III is dedicated to the description and modeling of investigator knowledge. Section IV discusses the potential attack scenarios reconstruction. It explains how states are represented and describes how hacking scenarios are inferred using forward and backward chaining. In Section V, a case study of a compromised FTP server is provided. Finally, the work is concluded in Section VI.

II. EXECUTION-BASED INVESTIGATION

After realizing that a security incident has occurred, an evidence can typically be seen as an undesirable state of the system. Starting with a partial description of the final system state (the word “partial” states that not all the system is accessible to assessment) and a description of the initial system state, which is assumed to be safe before the attack starts and known by investigators, we aim at generating the potential attack scenarios where hypotheses are automatically generated, if needed. Our approach considers an attack scenario as a graph where nodes are states through which the system progresses. The initial state denotes a safe state and the last state(s) may represent unsafe states. A final state satisfies the observed evidence. The arrows depicted in the graph link state s to state t when an action can be performed on s by the intruder to lead the system state to t . We consider an attack scenario as a composition legitimate and malicious actions.

a) Library of legitimate actions: Considering a digital attack scenario, it can be noticed that a large number of actions performed by intruders represents legitimate behavior on the system under investigation. Stopping a service, for instance, while having an administrative privilege, is a common task that could be performed by an administrator for maintenance purposes. However, such action plays an important role in compromising systems, if used with malicious intent. Since the number of legitimate actions is finite, specifying them within a library is feasible and helpful. A formal description of the system under investigation is sufficient to start building such a library.

b) A library of hypotheses: Specifying malicious actions in advance within a library may be problem because: a) the intruder behavior cannot be predicted in advance; and b) some hypotheses need to be advanced as there is always a lack of data on how the incident was conducted. A local privilege escalation action, for instance, requires to provide a hypothesis stating that one of the running services is locally vulnerable allowing the intruder to gain a privilege, if it is exploited. In this work, we found that in the majority of cases, the success of a malicious action is conditioned by a hypothesis that underlines its occurrence. While the description of a library of malicious actions is unpractical, we found that a library of hypotheses can be practical and supportive for at two reasons. First, the hypotheses are very seldom affected by the techniques, that attackers use, and type of the vulnerability. Second, there is a high correlation between a) a hypothesis and the system behavior at the moment a hypothesis is advanced; and b) a hypothesis and the remaining system behavior just after a hypothesis is generated. For instance, a hypothesis stating that a service has a vulnerability that could remotely grant a root privilege, may have negative effect on the system that can be described by an increase of the current user privilege.

III. KNOWLEDGE REPRESENTATION

Legitimate actions and hypotheses can be described using a knowledge base.

A. Legitimate actions modeling

A legitimate action is a pre-condition and an event. The pre-condition is represented by a predicate between state variables.

It should be satisfied in order to let allow the scenario fragment be executed. The event is the event that may occur and surely introduces changes to the system variables. It defines, for the set of variables that can be affected, a relation between their previous state and their new state. Actions can be collected by collaborating investigators to form a knowledge base of the legitimate actions.

To formally represent legitimate actions, we require defining a language that: a) satisfies the aforementioned representation; b) is expressive to be able to model complex systems and allow a better representation of the investigator knowledge; and c) is quite intelligible to be easily used by investigators which are not necessarily acquainted with the use of formal methods. We propose within this work to use the high-level formal specification language TLA^+ which was introduced for the specification of distributed and asynchronous systems [3]. The language provides a mathematical foundation for an effective reasoning, known as assertional reasoning. It is typeless and allows the description of states and state transitions. A state is a mapping from the set of all variable names to the collection of all possible values. A state transition is described by an action that is a relation between two states, a previous state and a new state, which are expressed using constants, variables and primed variables (the variable of the new state). Symbol $'$ is called the prime operator, it means, for example in the equation: $x' = x + 1$, that the variable x in the new state is 1 greater than in the previous state.

Now it can be easily seen that a scenario fragment as modeled in Section III-A matches well the form of a TLA^+ action. In fact, pre-conditions and events which represent the context of a scenario fragment can be described by state-predicates and relations between primed and unprimed variables.

B. Hypotheses modeling

We represent a hypothesis as a tuple containing three objects: a) the real assumption. It is similar to a TLA^+ state event except that variables that will be affected by the assumption (we call them hypothetical variables) should be different from the other variables used to describe legitimate actions. In fact, a hypothesis once stated, its change is constrained afterwards through all the attack progression. However, while it could be refined or generalized, it cannot substantially be changed; b) the prerequisite. It is optional and is a predicate that should be satisfied in order to let the hypothesis be selected; and c) the behavior. This is a constraint on a part of the system behavior, if the hypothesis is stated. It is a formula that should be kept true, having the form of a logical relation between the new state (further to the application of the hypothesis) and the previous state.

Given a hypothesis h that states that the FTP service has a local vulnerability that allows local users to gain an administrative privilege. From the attacker point of view, the user should be connected in order to let the hypothesis be applicable. In the other side, the system may behave, at the same time the hypothesis is stated, by enabling the user to escalate its privilege. Formally, the hypothesis is:

$$h\{ \text{Assumption } (ftpvul = local_root_pr), \\ \text{Prerequisite } (pr = 1), \text{Behavior } (pr' = pr + 1) \}$$

We denote by h^P and h^B the prerequisite and the behavior of the hypothesis h , respectively. Given a state s , $h^P(s)$ is equal True if h can be selected. Moreover, if the system is able to progress from state s to t , $h^B(s, t)$ should be equal True.

C. Representing inconsistencies

Accumulation of hypotheses during state progression needs care to avoid reaching a state under inconsistent hypotheses. Two forms of inconsistencies may arise. The first can be formally defined as a predicate containing hypothetical variable values. If the predicate is true for a state t , the set of system transitions on the way to that state should not be followed. The following predicate denotes two inconsistent hypotheses: a) the ftp service is vulnerable and b) the system is up to date:

$$\text{Inconsistency} \{ \begin{array}{l} \wedge \text{ ftpvul} = \text{"local_root_pr"} \\ \wedge \text{ systemstate} = \text{"uptodate"} \end{array} \}$$

The second form of inconsistency is related to hypotheses that use the same variables and whose intersection is equal to the empty set. For instance, the two hypotheses h_1 and h_2 which are equal to $x > 0$ and $x < 0$, respectively, are inconsistent.

D. Hypotheses arrangement

Hypotheses may depend on each other, even if they use different hypothetical variables. In fact, a hypothesis h_1 can be more specific or general than a hypothesis h_2 . Arranging hypotheses will be of high importance during scenarios reconstruction. In the following example, h_2 is More specific than h_1 . h_2 states that ftp service has a local vulnerability that grants root privilege is more specific than hypothesis h_1 which states that the *xinetd* service (which encompasses this transient FTP service) is vulnerable. Arrangement is represented as: $\text{Arrangement}\{h_2 \prec h_1\}$.

IV. A MODEL-CHECKER FOR HYPOTHESES-BASED INVESTIGATION

To automate the proof in the context of forensic investigation, we provide a model checking algorithm which generates the potential attack scenarios starting from a description of the system under investigation, available evidences, and a library of hypotheses. The generated graph provides a view of the states by which the system goes during the attack. It is built by ensuring that a given node is reached under consistent and optimal sets of hypotheses, while a straightforward reading of the node label indicates the alternatives under which a node is reachable.

A. Malicious action generation

Given a state s , we generate potential next states $T = \{\cup t_i\}$ such that every t_i is reachable from s , under the additional set of hypotheses H_{t_i} . Consequently, every t_i is reachable from the initial system state under $H_{t_i} \cup H_s$ (H_s is the set of hypotheses under which s is reachable). States in T are generated based on the fact that hypotheses in H_{t_i} dictate how system variables will be affected. Implicitly, for every pair of states (s, t_i) , we can generate an action A_i that allows the progress from s to t_i under hypotheses in H_{t_i} . Formally, for a variable v ($v \in \text{Var}$) we have: $s(v) \neq t(v) \Rightarrow \exists h \in H_{t_i}$ such that $h^B(s(v), t(v)) = \text{True}$. This means that, for every variable that is changed between s and t , there exists a hypothesis $h \in H_{t_i}$ that dictates how the variable is affected. We use $s \mapsto (t, H_{t_i})$ to say that state t is reachable from s under H_{t_i} .

B. Refinement of hypotheses:

Refinement will occur in two cases. The first case is a conjunction of two hypotheses that use the same hypothetical variables and whose intersection is different from the empty set. When the two hypotheses are joined together, they will be refined into their intersection. The second case is a conjunction of two hypotheses h_1 and h_2 , which do not use the same hypothetical variables and such that $h_2 \prec h_1$. Suppose that a hypothesis h_1 is advanced to reach state t from state s . As s is also reachable under hypothesis h_2 , the two hypotheses will be combined to the more general and t will be reachable under h_1 only.

C. State representation

A node s in the generated graph is represented in the form of *node core* and *node label*. The core of s , denoted by s^c , represents a valuation of the entire system variables. The label of s , denoted by s^l , represents the potential sets of hypotheses under which the node core is reached. Given a system described by variable x and a state s which represents a valuation of variable x ($x = 1$). We suppose that s can be reached under two possible sets of hypotheses $\{h_1 = 1, h_2 = 2\}$ and $\{h_1 = 3, h_3 = 3\}$. Instead of representing these two states using two different nodes in the graph, our representation will be in the form of $1\{(h_1 = 1, h_2 = 2), (h_1 = 3, h_3 = 3)\}$ where 1 is the node core, $(h_1 = 1, h_2 = 2)$ and $(h_1 = 3, h_3 = 3)$ are node environments, and $\{(h_1 = 1, h_2 = 2), (h_1 = 3, h_3 = 3)\}$ is the node label.

D. Potential scenarios reconstruction

A scenario reconstruction algorithm, which uses a forward and a backward chaining phase, employs three data structures \mathcal{G} , \mathcal{U}_F and \mathcal{U}_B . \mathcal{G} refers to the reachability directed graph under construction generated. \mathcal{U}_F and \mathcal{U}_B are FIFO queues, containing states whose successors have not being yet computed during forward and backward chaining phases, respectively. The algorithm assumes that \mathcal{A} is the library of legitimate actions, \mathcal{H} is the library of hypotheses, and EvState is a predicate characteristic of a terminal state (the digital evidence). To append a node to \mathcal{G} , the algorithm uses function $\text{Append}(\mathcal{G}, t, t \mapsto s)$ with a pointer to its predecessor state s . Besides, a state s with the set of hypotheses that make it reachable, are attached to a FIFO queue \mathcal{U} using function $\text{Append}(\mathcal{U}, (s, H_s))$ and detached using the function $\text{Tail}(\mathcal{U})$. The algorithm works using three phases:

Initialization phase: Graph \mathcal{G} and queues \mathcal{U}_F and \mathcal{U}_B are created and initialized respectively to empty set \emptyset and empty sequences $\langle \rangle$. Then, all the initial system states are computed and appended to \mathcal{G} with a pointer to the *null* state (no predecessor). During the latter step, every computed state is checked whether it satisfies the evidence predicate EvidenceState . In case of satisfaction, it is attached to queue \mathcal{U}_F ; otherwise, it is considered as a terminal state and appended to queue \mathcal{U}_B , in order to be retrieved in backward chaining phase.

Forward chaining phase: All scenarios that originate from the set of initial states are inferred in a forward chaining manner. This involves the generation of new hypotheses and evidences

that are related to these scenarios. During this phase, the algorithm starts with \mathcal{U}_F equal to the set of initial system states. Then, and until the queue becomes empty, state s (representing the tail of \mathcal{U}_F) is retrieved and its successors are computed using two different ways. In the first way, the algorithm generates the set $T = \cup\{t_i\}$ of states that follows s further to the execution of a legitimate action $A_{t_i} \in \mathcal{A}$. In this case, no new hypotheses are advanced to let the system move from s to t_i . Every state t_i is appended to \mathcal{G} as follows: If there is no state x in \mathcal{G} whose core x^c is equal to t_i , a new node (set to t_i) is appended to \mathcal{G} with a label equal to the set of hypotheses H_s under which node s was reachable, and a predecessor equal to s . Then, (t_i, H_s) is appended to \mathcal{U}_B , if state t_i satisfies the evidence predicate *EvidenceState*, otherwise it is attached to \mathcal{U}_F . If there is x in \mathcal{G} whose core x^c is the same as state t_i , then one can state that t_i has been added previously to \mathcal{G} . Thus, a pointer is simply added from x to its predecessor state s . In the second way, the algorithm generates the set of states and hypotheses $T = \cup\{(t_i, H_{t_i})\}$. $H_{t_i} \subset \mathcal{H}$ so that:

- The system is able to progress from s to t_i under set H_{t_i} .
- The set of hypotheses $H_s \cup H_{t_i}$ is consistent (inconsistency expression does not hold).

- The set of selected hypotheses is minimal, meaning that: a) there is no set of hypotheses H' that is an upper-set of H_{t_i} and that allows the system to progress from s to t_i similarly to H_{t_i} ; and b) there should be no hypothesis $h \in H_{t_i}$ that could be replaced by a hypothesis $h' \in H'$ such that $h' \neq h$, $h' \prec h$ and H' allows the system to progress from s to t_i as H_{t_i} does.

Every generated state t_i is appended to graph \mathcal{G} as follows:

- If there is no state x in \mathcal{G} whose core x^c is equal to t_i , a new node (set to t_i) is appended to \mathcal{G} with a label equal to the set of hypotheses $H_s \cup H_{t_i}$, and a predecessor equal to s . Then, tuple $(t, H_s \cup H_{t_i})$ is appended to \mathcal{U}_B if t satisfies the evidence predicate *EvState*, otherwise it is attached to queue \mathcal{U}_F .

- If there exists a node x in \mathcal{G} whose core x^c is the same as state t_i and whose label includes the set of hypotheses $H_s \cup H_{t_i}$ under which state t_i is reachable, then a pointer is simply added from node x to its predecessor state s_n .

- If there exists a node x in \mathcal{G} whose core x^c is the same as state t_i , but whose label does not include the set of hypotheses $H_s \cup H_{t_i}$, then the label of state x is updated as follows:

a) The set of hypotheses $H_s \cup H_{t_i}$ is added to $Label(\mathcal{G}, x)$, the label of state x .

b) Any environment from $Label(\mathcal{G}, x)$, which is a superset of another in this label, is deleted to ensure minimality.

c) Any set of hypotheses H from $Label(\mathcal{G}, x)$, for which there exists another set H' which is composed of minimal hypotheses (according to hypotheses arrangement) regarding set H is deleted to ensure hypotheses minimality.

d) In the case where the set of $H_s \cup H_{t_i}$ is still contained in the label of state x (i.e., not deleted in step (b)) then node x is pointed to s and node t_i is appended to \mathcal{U}_B if it satisfies the evidence predicate *EvState*; otherwise, it is attached to \mathcal{U}_F .

Every node label in the generated graph is provided with the following properties: a) soundness: a node x holds each environment E_i ; b) consistency: none environment E_i in the node label is inconsistent; c) completeness: every environment E in which a node x holds is a superset of some E_i ; and d) minimality: none environment E_i is a proper subset of any other.

Backward chaining phase: All optimal scenarios that could produce states satisfying the evidence predicate *EvState* generated in forward chaining phase, are constructed. This helps obtaining potential and additional scenarios that could be the root causes for the available evidences. During backward chaining, the algorithm starts with a queue \mathcal{U}_B holding the set of terminal states; the ones that satisfied the predicate *EvState* in forward chaining. Then, and until the queue becomes empty, tuple (t, H_t) representing the tail of \mathcal{U}_B is retrieved and the predecessor states of t (the ones that are not terminal states) are computed using two different manners. In the former, we take an interest to the generation of the set of states $S = \cup\{s_i\}$ that precede t if a legitimate action $A_{s_i} \in \mathcal{A}$ would have been executed from state s . In this case, no new hypotheses would have been advanced. A state s_i is appended to \mathcal{G} as follows:

- If there is no state x in \mathcal{G} whose core x^c is equal to s_i , a new node (set to s_i) is appended to the graph with a label equal to the set H_t under which node t is reachable. Then, a pointer is added from t to s_i and tuple (s_i, H_t) is appended to \mathcal{U}_B .

- If there exists a node x in \mathcal{G} whose core x^c is the same as state s_i , then a pointer is simply added from the node t to the predecessor state x .

In the latter case, the algorithm generates a set of tuple $S = \cup\{(s_i, H_{s_i})\}$. where $H_{s_i} (\subset \mathcal{H})$ is the set of generated hypotheses that would have allowed the system to move from s_i to t . Every generated state s_i is appended to \mathcal{G} as follows:

- If there is no state x in graph \mathcal{G} whose core x^c is equal to s_i , a new node (set to s_i) is appended to \mathcal{G} with a label equal to the set $H_t \setminus H_{s_i}$ (H_t minus H_{s_i}). Then a pointer is added from t to s_i and the tuple $(s, H_t \setminus H_{s_i})$ is appended to \mathcal{U}_B .

- If there exists a node x in \mathcal{G} whose core x^c is the same as state s_i and whose label includes the set of hypotheses $H_t \setminus H_{s_i}$ under which s_i is reachable, then a pointer is simply added from the node x to the predecessor state s_i .

- If there exists a node x in \mathcal{G} whose core x^c is the same as state s_i , but whose label does not include the set of hypotheses $H_t \setminus H_{s_i}$ under which state s_i is reachable, then the label of state x is updated as follows:

a) The set of hypotheses $H_t \setminus H_{s_i}$ is added to $Label(\mathcal{G}, x)$.

b) Any set of hypotheses from $Label(\mathcal{G}, x)$, which is a superset of other elements in this label, is deleted.

c) Any set of hypotheses H from $Label(\mathcal{G}, x)$, for which there exists a set H' which is composed of minimal hypotheses regarding set H is deleted to ensure hypotheses minimality.

d) If the set of hypotheses $H_t \setminus H_{s_i}$ is still contained in the label of state x then a pointer is added from node t to node x and node s_i is appended to \mathcal{U}_B .

V. CASE STUDY

We propose an investigation of a compromised FTP server. For the sake of simplicity, we model the system using three variables: *srv*, *FtpUsrLogin*, and *pr* which represent the FTP services status, users status (logged or not), and provided user privilege, respectively. The system runs initially with an activated FTP service ($srv[21] = "ftp"$, no user is logged in ($FtpUsrLogin = False$), and no shell is spawned to any user ($pr = 0$). The compromise is detected when the FTP service is found in a denial of service state, while no privilege is

granted. This can formally specified by: $EvState \triangleq (srv[21] = "null") \wedge (pr = 0)$.

A. Legitimate system description

For the sake of space reduction, we will only consider the set of actions that are part of the conducted hacking scenarios:

- *Remote User login*: A remote user logs in to FTP server and gets a simple user privilege.

$RmtUsrLogin \triangleq \wedge srv[21] = "FTP" \wedge usrlogin = "False" \wedge pr = 0 \wedge usrlogin' = "True" \wedge pr' = 1 \wedge UNCHANGED\ srv$

- *Remote User logout*: A remote user which is connected to FTP server logs out. Its privilege falls down to zero.

$UsrLogout \triangleq \wedge usrlogin = "True" \wedge usrlogin' = "False" \wedge pr' = 0 \wedge UNCHANGED\ (srv)$

- *Service stopping*: A privileged user which has previously logged in to FTP server stops the FTP service.

$SrvStop \triangleq \wedge srv[21] = "FTP" \wedge usrlogin = "True" \wedge pr = 2 \wedge srv' = [srv\ EXCEPT\ ![21] = "null"] \wedge UNCHANGED\ (usrlogin, pr)$

B. Hypotheses library

For the sake of space, we only focus on relevant hypotheses.

- *Vulnerable FTP service 1*: This states that FTP service has a vulnerability that can only be exploited locally. It lets a locally connected user escalate its privilege.

$h_1 \{ \text{Assumption } (ftpvul1 = local_root_pr), \text{Prerequisite } (pr = 1), \text{Behavior } (pr' = pr + 1) \}$

- *Vulnerable FTP service 2*: This states that FTP service has a remote buffer overflow vulnerability that, when exploited, grants a privileged shell while denying the service.

$h_2 \{ \text{Assumption } (ftpvul2 = DoS_local_root_pr), \text{Prerequisite } pr = 0, \text{Behavior } pr' = pr + 2 \}$

- *Installed rootkit*: This states that a rootkit is installed on the server and, once activated, it prevents the administrator from noticing that a FTP user has connected to the server.

$h_3 \{ \text{Assumption } (malicious_soft = "rootkit"), \text{Prerequisite } FtpUsrLogin = True, \text{Behavior } FtpUsrLogin' = True \}$

- *Vulnerable xinetd service*: This states that *xinetd* service is vulnerable. This vulnerability can only be exploited locally. It lets a locally connected user escalate its privilege.

$h_4 \{ \text{Assumption } (xinetdvul1 = local_root_pr) \text{Prerequisite } (pr = 1), \text{Behavior } (pr' = pr + 1) \}$

The arrangement $Arrangement\{h_1 \subset h_4\}$ states that hypothesis h_1 is part of hypothesis h_4 . In fact, the FTP service is encompassed in the *xinetd* service. Thus, a vulnerable FTP service implies a vulnerable *xinetd* service:

C. Potential scenarios generation

Figure 2 describes the results obtained. The state in bold circle represents the initial state, while states in double circles represent those satisfying predicate $EvState$. The evidence can be generated under two hypotheses: a) the running FTP service has a local vulnerability that allows a locally connected user to escalate its privilege, and b) the running FTP service has a remote DoS vulnerability that allows a remote user to gain privileged access while denying the service. Two possible scenarios

may be distinguished: a) An intruder logs in remotely as a simple user on the FTP server and then escalates its privilege by exploiting a local FTP vulnerability. Once its gains an administrative privilege, it stops the FTP service and logs out; and b) an intruder remotely exploits a DoS vulnerability that allows it to immediately gain administrative privilege on the server, while killing the FTP service. After that, it logs out.

Hypothesis h_4 is not chosen for moving the system from state 2 to state 3, since hypothesis h_1 is minimal than it. Moreover, we notice that after reaching state 6 under h_1 and h_3 , the system executes action *UsrLogout* and finds that s_7 is reachable under hypothesis (h_1, h_2) or (h_1, h_3) . Since $\{h_1\} \subset \{h_1, h_3\}$, $\{h_1, h_3\}$ is not appended to the label of state 6 and state 7 is not linked to its predecessor, state 6.

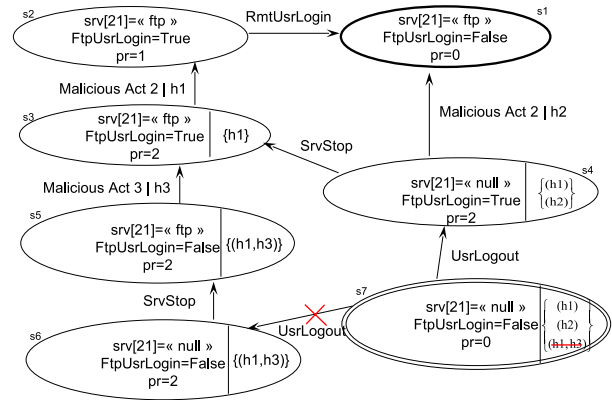


Fig. 1. Generated potential attack scenarios

VI. CONCLUSION

We proposed in this paper a novel approach for digital forensic investigation that aims at reconstructing attack scenarios using a library of legitimate actions and a library of hypotheses. The approach enables automated generation of hypotheses to fill in any potential lack of data on the malicious intruder behavior. The approach is provided with a model checking algorithm for an automated generation of potential attack scenarios.

REFERENCES

- [1] Christopher Elsaesser and Michael C. Tanner. Automated diagnosis for computer forensics. Technical report, The MITRE Corporation, 2001.
- [2] Warren G. Kruse II and Jay G. Heiser. *Computer Forensics : Incident Response Essentials*. Addison-Wesley Professional, 2001.
- [3] Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.
- [4] Slim Rekhis and Nouredine Boudriga. A Formal Logic-based Language and an Automated Verification Tool For Computer Forensic Investigation. In *Proceedings of the 20th ACM Symposium on Applied Computing (SAC 2005)*, 2005.
- [5] Slim REKHIS and Nouredine BOUDRIGA. A Temporal Logic-based Model for Forensic Investigation in Networked System Security. In *The Third International Workshop "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-05)*, pages 325–338. Springer Verlag, LNCS 3685, September 2005.
- [6] Tye Stallard and Karl Levitt. Automated analysis for digital forensic science: Semantic integrity checking. In *Proceedings of the 19th Annual Computer Security Applications Conference*, 2003.