Proceedings of the 2001 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 5–6 June 2001

Design and Implementation of Property-Oriented Detection for Link State Routing Protocols

Feiyi Wang, Fengmin Gong, Felix S. Wu, Hairong Qi

Abstract— **This paper presents a new intrusion detection approach, property-oriented analysis and detection (POD). We discuss both the generic paradigm of this new approach and our design and implementation experience within the context of link state routing system. A routing system is modeled as a set of distributed processes. A *property* is defined as a *state predicate* over system variables. For link state routing protocol, the overall converging property $\mathcal{P}$ is defined as the "equality" among routing information bases maintained by all processes. We decompose routing protocol into different computation phases and specify them using generic Input/Output Automata (IOA). For each phase, we use state predicates (liveness and safety property) as our guide for observation and analysis. The goal of the detection algorithm is to construct a validation path based on the history to determine if the fault is *natural* or *malicious* when the property $\mathcal{P}$ is rendered invalid by faults. The contribution of this paper is three-fold: First, this new detection paradigm proposed differs from the traditional signature based or profile based intrusion detection paradigms in the sense that it utilizes stable property as a starting point, and correlates the history and future to validate changes caused by natural faults and identify the malicious faults in the system; Second, by exploring primary concerned system properties, we show that detection effort can be conducted in a more focused and systematic fashion. Third, our design and implementation experience shows that how this approach can be effectively applied in complex distributed system, i.e., link state routing system.**

Keywords— **Intrusion detection, distributed system monitoring, security management**

I. Motivation and Related Work

INTRUSION detection (ID) as a second line of defense is in no doubt attracting more and more research attention these days. The basic approaches of ID can be generally classified as either signature-based or statistical-based. Signature-based detection relies on its understanding of the nature of the attack and identify a particular attack by its state transition sequences or patterns. Statistical-based detection bases on the comparison of *past-observed-normal* and *under-or-after attack* statistical profile.

Our previous work has been focusing on vulnerability analysis for link state protocol [1], [2], building both statistical-based [3] and finite state machine-based [4] detection engine under JiNao [5], [6] project framework. We also explored the effective network management based approach to do similar intrusion detections [7], [8]. Some of these traditional techniques including finite state based and statistical-based approaches used in our system have been studied in other context for intrusion detection as well [9]. Our experience reveals similar problems that are faced by other researchers, some of which are amplified by the unique characteristics of routing system. These problems are summarized next.

Most of the previous intrusion detection approaches are based on certain attack models, which are developed with the understanding of the vulnerabilities of the system from attacker's point of view: (1) where are the weak points? (2) how can one exploit and compromise the system? One way of doing vulnerability analysis is to check system's operational behavior against possible attacker's cause of actions. This approach is difficult because it needs good models for both system itself and the potential attackers. Routing system as a distributed concurrent system can exhibit a wide variety of possible behaviors. Reasoning directly about these behaviors can become quite complex, with many different cases to consider. Moreover, as an intelligent entity, an attacker can behave in a highly unpredictable fashion. It seems that any profiles established for them can be circumvented one way or the other given the reasonable understanding of basic techniques used to profile them. It is not clear if there are satisfactory methods to cope with this complexity.

This uncertainty of potential vulnerabilities and complexity of system behavior directly impacts the corresponding detection algorithm. For example, the finite state machine-based (FSM) approach adopted in our previous JiNao project captures the "signatures" of attacks we can deduce based on the case analysis on system and attacker's behaviors. However, it is hard to assess the detection coverage because each attack signature has to be discovered and programmed into the engine. Overall, there is more

Feiyi Wang is with Advanced Network Research Group, MCNC, Research Triangle Park, NC. Email: fwang2@mcnc.org

Fengmin Gong is with Intrusion Detection Technology Division of IntruVert Network Inc. Email: fengmin@intruvert.com

Felix S. Wu is an associate professor in University of California, Davis, Email: wu@cs.ucdavis.edu

Hairong Qi is an asssitant professor in University of Tennessee, Knoxville. Email: hqi@utk.edu

This work is sponsored by U.S. Department of Defense Advanced Research Projects Agency and U.S. Air Force Rome Laboratory under contract F30602-96-C-0325.

ISBN 0-7803-9814-9$10.00

of art than science for the approach. To overcome the complexity and uncertainty encountered in our previous studies, a new detection algorithm is presented in this paper. It bases upon two fundamental ideas: First, for any distributed systems, even though the system operational behavior can be quite complex, it is possible to focus on the critical properties that concern us the most in order to simplify and organize the analysis process; Second, in order to validate the current system state, it is necessary to correlate system history. This idea reflects the basic philosophy that *one cannot understand the present without learning the history*. These two ideas are closely related to each other. The former (property analysis) provides the foundation, while the latter step is an instantiation toward constructing the detection algorithm.

The rest of the paper is organized as follows. Section II describes the general paradigm of POD approach and basic steps to apply it in a routing system. Section III presents a summarization of property analysis for link state routing system.[1]. Section IV-A discusses various design and implementation issues related to POD algorithm in GIANT (Global Intrusion Assessment Through Distributed Decision Making) framework. Section V analyzes our experimental results and make conclusion remarks.

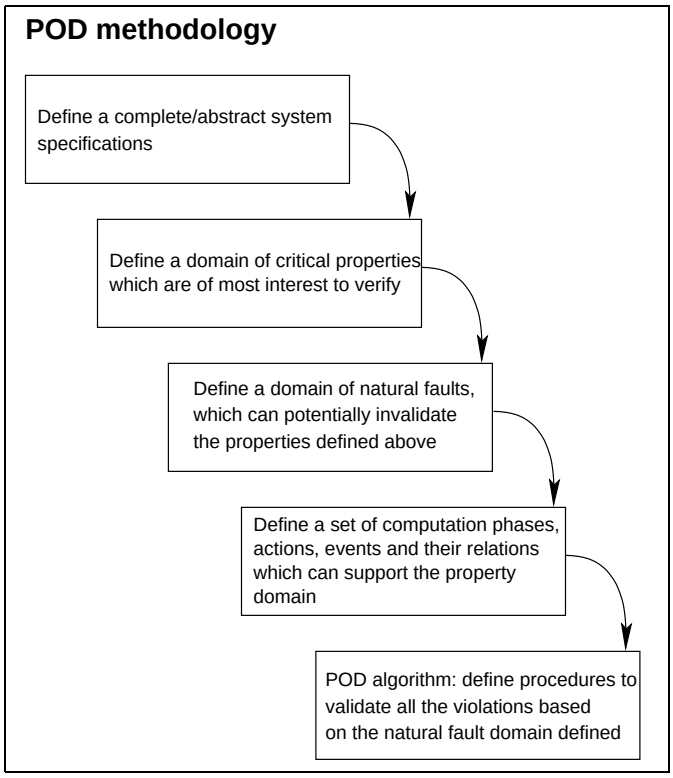

Fig. 1. Illustration of general POD methodology

II. Property-based Analysis and Detection

A. Generic Paradigm of POD Approach

In general, if we can *completely* specify a system's properties, also assume that it is possible to monitor and verify these properties at run time, then one can claim to able to able to detect all the faults. However, it would not be practical for any complicated system. A natural solution would be to focus on critical properties of the system which concern us the most. Therefore, the challenges are what properties should we consider as critical properties and how do we verify that a system does hold these properties.

We first consider a simple example and then outline the general steps to take for applying the POD in another domain. Suppose we want to do intrusion detection for a banking account, which has \$1,000 balance. To further simply the exposition, we assume that there is no withdrawal, no deposit and no interest at all. It is evident to see that $x = \$1,000$ is the critical property for the system: any time this property is violated, we know that some thing goes wrong and an alarm should be triggered. Figure 1 illustrates the general steps to take when applying this approach.

The basic steps to be performed are: (1) We should have a good specification of the target system. This step is critical since all properties will be derived and proved from it. (2) A complex system often exhibit a variety of properties, we must clearly define a domain of "critical property" that we are mostly concerned to monitor and verify. Domain of interest can certainly grow with more understanding of system itself. However, the boundary is vital since this dictate the detection coverage of the ID system. (3) We need to inspect if there are inherent natural faults inside the system which can invalidate the defined properties. In our simple banking example, there is not such a natural fault set, i.e., any faults which cause $x \neq 1000$ should be considered a compromise of the system. However, as we can see later, such natural faults could be quite common in a routing system. (4) We need to define a set of phases and procedures which make the system uphold these properties. (5) A detection algorithm is in essence a process of verifying these phases and procedures are faithfully followed by examining and correlating both the history and potential future traces. Much more details on how this general paradigm applied and implemented in practice will be discussed in later sections.

B. POD Methodology Applied in Link State Routing

To apply the above POD methodology to a complex system such as link state routing, we first make the following observations: (1) all the vulnerability analysis and attacks we have come up so far, they all eventually manifest some illegal changes to the routing information base. (2) In link state routing protocol, routing information base is essentially a topology database, which does not change very fre-

[1]Limited by space, we omit the IOA specification and proof part, interested readers can refer to [10] for more details

quent in normal situations. (3) these routing information base information is readily available to an outside observer. With these observation in mind, we chose synchronization of routing information base as the critical property for link state routing.

Link state routing has one important characteristic that each process' view of network topology is complete and constructed by a set of independent LSAs. These LSAs describe neighbors they are connecting to and states maintained by this process. When the network is stable (stable in the sense that no fault such as link failure occurs), the view of network topology by each process should be synchronized. This characteristic motivates us to associate it with liveness property of "something good" (routing information base is synchronized) *eventually* happens. The properties can be satisfied if "nothing bad" (natural or malicious faults) happens. We make more specific definitions as follows:

A *routing system* $\mathcal{R}$ is modeled by a set of processes p_1, p_2, ..., p_n, each of which is associated with a set of local program variables. Each process possesses a meta variable called *routing information base* (RIB) that consists of a set of LSAs. Let $\mathcal{P}$ be a *state predicate* defined over variables of $\mathcal{R}$, that is, $\mathcal{P}(S)$ is either true or false for a global state S of $\mathcal{R}$. The predicate $\mathcal{P}$ is said to be a *stable property* of $\mathcal{R}$ if $\mathcal{P}(S)$ implies $\mathcal{P}(S')$ for all global states S' of $\mathcal{R}$ reachable from the global state S of $\mathcal{R}$. Examples of stable properties in traditional environment are "computation has terminated", "the system is deadlocked", etc. In the context of our discussion, the stable property $\mathcal{P}$ of routing system is defined as:

$$\mathcal{P} \triangleq p_1(RIB) = p_2(RIB) = \ldots = p_n(RIB)$$

Strictly speaking, the stable property $\mathcal{P}$ is not stable once a fault, such as link failure, occurs. Just like "the system is deadlocked" is not stable any more if the deadlock is "broken" and the computation is re-initiated [11]. To keep the exposition simple, we divide a system computation into several phases. Each phase has the characteristics that: (1) reaching stable property represents the termination of one particular phase, and (2) deviating from stable property starts a new phase. The key point is that $\mathcal{P}$ defined here provides us a starting point for observation. There *must* be a trigger event which renders the system unstable. One might or might not be able to observe such a trigger event, but the most important effect one will observe is: $\mathcal{P}$ is invalid. Our detection algorithm starts from this point to construct a validation path. A validation path is essentially a correlation process which can correlate the past events (history) to validate if the *previous* computation phases are followed according to the specification and *their* function properties are satisfied. If such validation does not hold, it is deemed as an intrusion event and the corresponding alarm should be triggered.

As mentioned above, we are reducing the task of validating history by validating the properties. Therefore, the critical properties defined have a direct impact on the validation process. From the wisdom of distributed program analysis, we find that there are two kinds of prominent properties which are capable of capturing and characterizing the distributed behavior – safety property and liveness property. The safety property has the form of "some things are not allowed to happen". It is analogous to partial correctness and expressed by invariant assertions which must be satisfied by the system state at all times. Liveness property has the form of "some things will happen." Examples include termination requirements in sequential programs. Safety and liveness properties are useful guidelines for helping us establish validation path for each phase. Note that even though theoretically it is possible that all properties can be expressed directly using assertions [12], we are not attempting to do so due to vast amounts of states for any practical protocols. The properties we propose and thereafter prove are the primary function properties that are supported by protocol specification and that are sufficient to guarantee the convergence property $\mathcal{P}$.

To realize the above idea in a practical link state routing system such as OSPF, there are two major steps we have to take:

- An abstract routing system based on OSPF protocol needs to be specified. The goal is to capture the essence of link state routing and keep the important details visible, while at the same time simplify other insignificant aspects. The simplified system provides the necessity of giving more precise specification of protocol behavior. Partitioning system into multiple computation phases further reduces the difficulties of tackling such a complicate distributed system. To specify each computation phase of OSPF, we use IOA for formalization, and prove that their properties are valid and supported by the specification. (Section III)
- A detection algorithm that constructs the validation path is then developed based on both the protocol specification and its corresponding critical properties for each phase.

We also consider the various aspects of applying this new POD approach in a practical context. This is presented in Section IV-A. We further take examples of malicious attacks developed in our previous work to illustrate how the detection algorithm works.

III. Link State Routing Property Analysis

A. Partition and Abstraction

As an Internet standard, OSPFv2 specification [13] includes many details which are not essential for our discussion. The procedure we presented simplifies it in three aspects: (1) OSPF allows sets of networks to be grouped together and such a group is called an area. Therefore, routing in OSPF can take place in two levels: inter-area routing and intra-area routing. Although this information

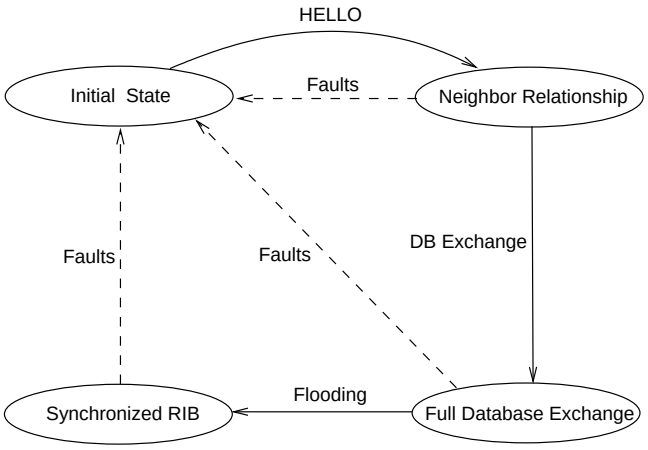

Fig. 2. An abstraction of state transition

hiding enables a significant reduction of routing traffic in practice, we only consider flat area routing, i.e., backbone routing; (2) OSPF considers many different network types on top of which it is running, and has corresponding mechanisms to best utilize it. For example, it reserves multicast address `224.0.0.4` and `224.0.0.5` for establishing neighbor relationship on a broadcast network such as Ethernet. Here, we consider primarily point to point network; (3) To inter-operate with other routing protocols such as BGP, OSPF defines AS-external-LSA to describe the routes not originated from the domain, which we consider inessential in our discussion.

Based on this simplification, we partition OSPF routing protocol into **Hello**, **DB exchange** and **Flooding** phases, an abstraction of state transition driven by these three phases is illustrated in Fig. 2. In the rest of this section, we introduce basic notations for routing protocol specification, then give an abstract algorithm of LSRP based on the three-phase partition. We argue that LSRP algorithm can maintain up-to-date topology information.

We use p, q, r to denote individual process, and *dot concatenation* to represent data structures associated with that process. For example, $p.RIB$ and $p.LSA$ denote the p's routing information base and an LSA in the $p.RIB$ respectively. We use LSA_i^j to distinguish the LSA's originator i and the neighbor j who relays this copy of LSA over during the flooding phase. When the context is clear, we skip the *process* prefix for the associated data structure we are referring to. $p.nbrS$ is an array of *neighbor states*, maintained by process p for each of its neighbors. It records what process p *thinks* what its neighbor state should be. This neighbor state of i satisfies: $p.nbrS.[i] \in \{Down, Init, ExStart, Exchange, Loading, Full\}$. Another intermediate state $2way$ is skipped because this state indicates that adjacency should not be established. Within our domain of discussion, all processes are valid to establish the adjacency, so it is safe to ignore the state. In our specification, the sequence is *totally ordered* with respect to relation $<$. For example, $p.nbrS[i] \leq ExStart$ implies $p.nbrS[i]$ can be any one of $Down$, $Init$, and $ExStart$. We use $p.nbrO$ and $p.nbrA$ to denote the set of passive outgoing neighbors [2] of p and active neighbor set of p. $p.nbrO$ is usually the *a-priori* knowledge of p, while $p.nbrA$ is established dynamically, $p.nbrA \subseteq p.nbrO$. The only communication primitives used in our specification are *send* and *recv*. Before we go into details on each computation phase's algorithm, a high level link state routing algorithm is presented below.

Algorithm 1 LSRT: process i's algorithm

```
1:  var i, j, k: process id
2:  var m, m': local topology view
3:  var nbrO: passive neighbor set, pre-configured
4:  var nbrA: active neighbor set, initially empty
5:  repeat
6:  Once i becomes operational:
7:    for each neighbor j ∈ nbrO do
8:      H(i, j)
9:    end for
10:   for each neighbor j ∈ nbrA do
11:     DB(i, j)
12:   end for
13: Once i's local view changes:
14:   if link ⟨i, j⟩ becomes operational then
15:     H(i, j)
16:     DB(i, j)
17:   else
18:     m := changed local view
19:     F(j, m)
20:   end if
21: Once i receives new updates from neighbor j:
22:   m' := new updated view
23:   F(i, m')
24: until no more new updates
```

$H(i, j)$ denotes Hello phase which is to establish the neighbor relationship between processes i and j. If the step is successful, then $nbrA := nbrA \cup \{j\}$. In this sense, $H(i, j)$ is a session management protocol, responsible to open and close a session between two neighbor processes. All subsequence information exchange is based on the assumption that there is open session in between. $DB(i, j)$ denotes DB exchange phase which is to exchange complete RIB between i and j. $DB(i, j)$ phase is called when there is new link or new process coming up. It is particularly helpful to expedite database convergence when two disconnected networks

[2] passive outgoing neighbors is defined as a pre-configured knowledge on the possible neighbor processes, which is a equal or subset of physical neighbor processes

reconnect. $F(i, j, m)$ denotes the flooding step, whenever new update (other process' local view changes) received or self's local view changes, this phase will be called for propagating these updates. If we assume that flooding rules can guarantee that any new update information will reach every connected process, then it is evident that the algorithm has the property: *Assume that the topological changes cease at some time, then, eventually, every process in the network knows the correct topology of its connected component.* This property can be proved by induction on the distance from every node. The base case follows the assumption that each process knows its local topology status. The induction follows from the one-way nature of the flooding. Subtle issues related to session management and flooding will be discussed in the following sections.

B. Property Specification and Analysis

In this section, we summarize properties defined for each computation phases discussed above, with particular emphasis on flooding phase since it plays such a crucial role in link state routing system. Formally, the correctness of these properties can be validated through IOA specification.

Property 1 *Hello phase satisfies: for any two process i and j:*

$$alive(i) \wedge alive(j) \wedge alive(\langle ij \rangle) \text{ leads to } (i.nbrS[j] = ExStart) \wedge (j \in i.nbrA) \quad (1)$$

Property 2 *Hello phase satisfies: for any two process i and j:*

$$alive(i) \wedge (\neg alive(j) \vee \neg alive(\langle ij \rangle)) \text{ leads to } i.nbrS[j] = Down \wedge j \notin i.nbrA \quad (2)$$

Property 3 *Hello protocol guarantees that the neighbor relationship is bidirectional:*

$$RID(j) \in i.nbrA \text{ leads-to } RID(i) \in j.nbrA. \quad (3)$$

Property 4 *For any two processes i and j, without losing generality, if during the ongoing DB exchange session between i and j, i does not receive any new LSA updates, then:*

$$i.nbrS[j] = Full \Rightarrow i.RIB \subseteq j.RIB \quad (4)$$

If j at the same time does not have new LSA received either, then we claim that DB exchange will leads-to

$$i.nbrS[j] = Full \wedge\ j.nbrS[i] = Full \Rightarrow i.RIB = j.RIB \quad (5)$$

Routing protocol should keep only the most up-to-date information in the database, Therefore, when a routing process *thinks* it possesses a piece of new information and decides to flood it (in this case, a LSA) to the whole routing domain, one of the key issue is: how do we judge the success of this flooding? i.e., desirable properties. Without loss of generality, we formally define this property using the state variables defined above and the flooding algorithm establish the truth of $DFP(LSA)$,

$$DFP(LSA) \triangleq \forall i : alive(i) \Rightarrow newer(i.LSA, LSA) \vee equal(i.LSA, LSA) \quad (6)$$

, where i represents process, LSA denotes the new information being flooded, $i.LSA$ represents the corresponding information i already have (If i does not have this information at all, we regard $i.LSA$ as NULL). *equal* and *newer* are assumed functions that can compare the freshness between two LSAs. With this global property in mind, we further define truth that should be established from each individual process's perspective:

$$DFP(i, LSA) \triangleq \forall j : j \in i.nbrA \Rightarrow newer(j.LSA, LSA) \vee equal(j.LSA, LSA) \quad (7)$$

where the formula basically says for each individual process, the truth of successful flooding is that it has made effort that each of their functional neighbors to have either this new information, or newer information. With an extra requirement that all functional process are connected, it is not hard to come to this conclusion:[3]

Property 5 *If each process i establishes the truth of $DFP(i, LSA)$, then the routing system therefore establishes the truth of $DFP(LSA)$*

IV. POD Detection Algorithm and Its Implementations

The property oriented detection algorithm (POD) is based on the assumptions that every process holds a routing information base (RIB) which consists of a set of (all) non-faulty LSAs when the system is in stable state. We treat such a stable state $\mathcal{P}$ as an Init state for the detection algorithm. Also, we assume that a potent observer p_0 is doing *continuous* observation on all n processes.

One observation on all the routing attacks we have analyzed before is, to affect routing behavior, it boils down to force an unexpected change in routing information base (RIB). In turn, a change on RIB will reflect a change on one or more LSAs, which are a set of descriptions on link status. If we have such potent observer, then we can observe the first time when RIB changed. In reality, we have to rely on either the real time inquiry or the traces generated by running process, which is the topic of Section IV-A. The detection algorithm takes the advantage of this clear starting point and then enumerates all possible cases on

[3]For more details on proof, please refer to [10], [14].

this LSA. For each possible case, we define its correlation actions based on the following principles: (1) by looking at the relevant process' history to see if we can construct a validation path which leads to the shift from the stable property $\mathcal{P}$. Although there may be vast amounts of details constituting to the *history* of these processes, our attention is on the properties defined for each computation phase. It is our belief that these properties characterize each phase and simplify the validation process (2) we also need to inspect the future, which should be a *converging process*, i.e., the system should (under normal situations) converge back to the stable state $\mathcal{P}$. For a formal specification of this algorithm, please refer to [10] and [14]. In the following sections, we focus on discussing its implementation considerations and trace analysis.

A. POD Implementations

The overall property $\mathcal{P}$ is synchronization of RIB. We assume that routing system starts with a "clean" state, meaning that at some point after initialization, RIB is stabilized and therefore we can find a synchronized snapshot of it. To realize this in practice, we have to be able to have access to RIB. If we have the source code, it is easy to adapt it to dump RIB whenever it changes. Routing software such as `GateD` even provides a private interface for you to inquiry on RIB, which makes it possible to periodically "poll" the information out. Each RIB usually consists of quite a few LSAs and each LSA consists of quite a few links. If we do comparison based on each data field, it can be very time consuming. In practice, we employ a technique called "checksum comparison". This technique computes each RIB with same checksum algorithm and compare only the result of checksum. Algorithms such as MD5 possesses the following property: mathematically, there is little chances that two difference input messages can have the same checksum. There are cases that RIB's get partially synchronized in the sense that some parts of topology converge faster, each RIB of process is equal at some instance, then the synchronization is broken by new updates. This is not a concern of POD detection algorithm since at this stage, the system is assumed clean and the updated should be normal. POD should validate the changes as usual and update its synchronization point correspondingly.

An implicit assumption is that network topology will cease the changes in finite time so that RIB will eventually converge and the frequency of the changes should be able to be handled by validation process. Our observation on test-bed is that link and process failures are indeed rare in a medium sized environment (10-50 routers). Depending on how many neighbors one router is physically connected, the average update of RIB (with three neighbors) is 10 minutes, and converging process usually takes seconds. This is radically different from backbone (inter-domain) routing, where the routing databases with hundreds of thousands entries are constantly changing and updating. For a single route to converge, it can take 5 to 30 minutes. POD in its pure form is not suitable for this environment.

The steps POD performs for each RIB changes are based on algorithm presented on Page 94. Each checking involving either H and DB, or H, DB, and F, or F phase alone. In each case, the properties checked are sufficient to guarantee the correct converging behavior for the involved processes. We further details on how to validate properties for each phase.

In Hello phase, we have three properties to consider. Property (1) claims the neighbor establishment requirement: if two neighbor process and the link between them are operational, then the neighbor state *nbrS* should eventually transit from *Down* to *ExStart* and the neighbor ID will be added to set of *nbrA*. Property (2) claims that maintenance requirement: if one process is operational but its neighbor or link between them are not operational, then neighbor state *nbrS* should eventually transit to state *Down*; Property (3) claims the neighbor relationship is bidirectional, i.e., eventually, both neighbor IDs must exist in both processes' *nbrA*. Property (1) should be validated when the process starts up. Property (3) must be satisfied during the operation of process and property (2) should be validated when link is removed from the RIB. Since Hello message is sent periodically and the message body includes the current neighbor ID list, there is no problem for an observer to validate these properties.

In DB exchange phase, we have one primary desired property and two sub-properties to consider. Property (4) and (5) claim that if process i does not receive any new updates during the execution of DB exchange, then $i.RIB$ is a subset of $j.RIB$; if both processes i and j do not receive any new updates during the execution of DB exchange, then $i.RIB$ equals to $j.RIB$. To reach this (partial) synchronization point, there are two sub-properties to be satisfied. Property D_1 claims the two processes in DB exchange must establish the *Master/Slave* relation and a initial sequence number must be decided by *Master*; Property D_2 claims two processes must fully exchange RIB summary information and establish the difference set *LSR*. DB phase only happens when a process or link just becomes operational. When DB exchange only brings partial synchronization, a Flooding phase must be followed to fully synchronize RIB. The status of RIB can only be observed and validated by the runtime dump, *Master/Slave* election process can be observed and validated by the DB type message exchanged.

In Flooding phase, we defined one overall desired property (6) and one property (7) which needs to be satisfied by each individual process. It is not difficult to validate the second property for each individual process. However, it is more important to validate "when" should flooding procedure are called for and "what" should the procedure

TABLE I
MaxSeq Attack: First synchronized RIB snapshot

152.45.4.208	152.45.4.210
<rib lclock="59" reporter="152.45.4.208" LSA_num="3" RIB_checksum="30243" <nbr id="152.45.4.210" state="FULL" /> <nbr id="152.45.4.207" state="FULL" /> <lsa originator="152.45.4.208" link state ID="152.45.4.208" num_of_links="2" relay_rid="152.45.4.208" lsa_age="0" lsa_chsum="10336" lsa_seqno="2" <link linkid="152.45.4.210" linkcost="2" /> <link linkid="152.45.4.207" linkcost="5" /> </lsa> <lsa originator="152.45.4.210" link state ID="152.45.4.210" num_of_links="2" relay_rid="152.45.4.210" lsa_age="0" lsa_chsum="10594" lsa_seqno="2" <link linkid="152.45.4.208" linkcost="1" /> <link linkid="152.45.4.207" linkcost="1" /> </lsa> ... />	<rib lclock="60" reporter="152.45.4.210" LSA_num="3" RIB_checksum="30243" <nbr id="152.45.4.208" state="FULL" /> <nbr id="152.45.4.207" state="FULL" /> <lsa originator="152.45.4.210" link state ID="152.45.4.210" num_of_links="2" relay_rid="152.45.4.210" lsa_age="0" lsa_chsum="10594" lsa_seqno="2" <link linkid="152.45.4.208" linkcost="1" /> <link linkid="152.45.4.207" linkcost="1" /> </lsa> <lsa originator="152.45.4.208" link state ID="152.45.4.208" num_of_links="2" relay_rid="152.45.4.208" lsa_age="0" lsa_chsum="10336" lsa_seqno="2" <link linkid="152.45.4.210" linkcost="2" /> <link linkid="152.45.4.207" linkcost="5" /> </lsa> ... />

flood. The flooding validation is more related to overall protocol execution. As we discussed above, if DB phase can only establish the partial synchronization, a flooding phase should be followed. This is essentially the case where new LSU update received. There are four cases where a process will perform flooding: (1) local topological view changes. This includes a new neighbor process starts up or a new link becomes operational. The updated LSA is a self-originated with sequence number increased by one. This new LSA needs to be flooded to each process in *nbrA*. (2) a new LSA update received but not originated by itself or the received LSA with a *MaxAge*. This new LSA needs to be flooded to every process in *nbrA* except the incoming neighbor process. (3) a new self-originated LSA update received. A newer LSA with sequence number increased by one needs to be flooded to every process in *nbrA*. (4) An LSA in its own RIB reaches *MaxAge*. This LSA needs to be flooded to every process in *nbrA*. The flooding phase is validated by mainly observing incoming and outgoing link state update.

In the case where multiple topology changes happen to the RIB at the same time, this will increase the implementation difficulty. However, POD algorithm still applies without need of modification since it tracks the phases for each LSA changes and treats the multiple changes as seperate runs.

V. Experimental Study of POD

In this section, we discuss a case of malicious faults against POD detection algorithm. We use `MaxSeqNumber` attack developed in [2] as an example against POD algorithm. In this scenario, attacker forges an LSA instance in the following way: (1) it sets the LSA sequence number to `0x7FFFFFFF`, i.e., `MaxSeqNumber`; (2) it re-computes

ISBN 0-7803-9814-9$10.00

TABLE II
MaxSeq Attack: First RIB changes detected at 210

152.45.4.208	152.45.4.210
<rib lclock="59" ... /> <send lclock="93" reporter="152.45.4.208" to="152.45.4.210" pktype="LINK STATE UPDATE" /> <send lclock="94" reporter="152.45.4.208" to="152.45.4.207" pktype="LINK STATE ACKNOWLEDGEMENT" /> <rib reporter="152.45.4.208" LSA_num="3" RIB_checksum="30243" <nbr id="152.45.4.210" state="FULL" /> <nbr id="152.45.4.207" state="FULL" /> <lsa originator="152.45.4.208" link state ID="152.45.4.208" num_of_links="2" relay_rid="152.45.4.208" lsa_age="0" lsa_chsum="10336" lsa_seqno="2" <link linkid="152.45.4.210" linkcost="2" /> <link linkid="152.45.4.207" linkcost="5" /> </lsa> ...	<rib lclock="91" reporter="152.45.4.210" LSA_num="3" RIB_checksum="30116" <nbr id="152.45.4.208" state="FULL" /> <nbr id="152.45.4.207" state="FULL" /> <lsa originator="152.45.4.210" link state ID="152.45.4.210" num_of_links="2" relay_rid="152.45.4.210" lsa_age="0" lsa_chsum="10594" lsa_seqno="2" <link linkid="152.45.4.208" linkcost="1" /> <link linkid="152.45.4.207" linkcost="1" /> </lsa> <lsa originator="152.45.4.208" link state ID="152.45.4.208" num_of_links="2" relay_rid="152.45.4.208" lsa_age="0" lsa_chsum="10209" lsa_seqno="2147483647" <link linkid="152.45.4.207" linkcost="6" /> <link linkid="152.45.4.210" linkcost="3" /> </lsa> ... />

both the LSA and OSPF packet checksums. Then this forged packet is re-injected into the system. This attacking LSA instance will be considered the "freshest" by other processes because it has the maximum LSA sequence number. Eventually it will be propagated to the originator of this particular LSA. The originator, according to the OSPFv2 specification, "should" first purge the LSA instance (setting `MaxAge`) and then flood a new LSA carrying correct link status information and the smallest sequence number: `0x80000001`. Without this purging, all the processes (except the originator) will accept the faulty LSA instance. The newly issued LSA from the originator will have `0x80000001` as the sequence number which will be considered as the oldest and be discarded. A previous version of routing software we tested did not handle this `MaxAge` correct, namely the purging of `MaxSeqNumber` LSA is not done right. This fault (a combined program implementation bug and malicious attack) renders the illegal `MaxSeqNumber` LSA staying in every router's RIB for up to one hour until it reaches `MaxAge`. Potentially, this faulty LSA can cause faulty route calculation.

Table I and II shown here are actual traces from a fully-connected three nodes network which is part of the GI-ANT testbed. The trace format is from the XML DTD defined in last section. Their IP address are `152.45.4.207`, `152.45.4.208`, `152.45.4.210` respectively. Table I shows the first synchronization point: each process has different logical clocks but with the same checksum as 30243 (decimal). Looking further, we can compare each LSA inside the RIB, they are and should be the exactly the same. The right column in Table II shows the first point we detect the change of RIB: in this case, it is a new *LSA instance* with `MaxSeq` number and changed link cost value. Following the detection algorithm, we first identify the originator of this LSA, which is `208`. Then we check if Flooding properties are satisfied by the originator. To satisfied flooding prop-

erty, 208 must broadcast the local changed new view to all its neighbors. As obvious as it is, this type of link attacker can not replicate the flooding history as long as 208 is not compromised itself. This can be seen from the left column of the trace file. The validation process will fail and raise the alarm.

VI. Summary

In summary, we presented design and implementation of property oriented fault detection approach (POD) within the context of link state routing. The goal of POD is to construct a history and/or future validation path to validate the changes of stable property $\mathcal{P}$, which can be caused either by natural or malicious faults. As a methodology, POD can be generalized in other application domain such as ongoing BGP security project [15]. We believe that POD not only provides an interesting new angle of studying the vulnerability and intrusion detection of link state routing protocol, it also delivers a framework, in which both the vulnerability and detection can be conducted in a more focused and systematic way.

References

[1] F. Wang and S. F. Wu, "On the vulnerabilities and protection of OSPF routing protocol," in *IEEE 7th International Conference on Computer Communication and Network (IC3N)*, October 1998.

[2] B. M. Vetter, F. Wang, and S. F. Wu, "An experimental study of insider attacks on the OSPF routing protocols," in *The 5th IEEE International Conference on Network Protocols (ICNP)*, Atlanta, GA, October 28-31 1997, pp. 293–300.

[3] D. Qu, B.Vetter, F. Wang, R. Narayan, S. Wu, Y. Jou, F. Gong, and C. Sargor, "Statistical-based intrusion detection for OSPF routing protocols anomaly detection for link state routing protocols," in *The 6th IEEE International Conference on Network Protocols*, Austin, Texas, October 13-16 1998, pp. 62–70, IEEE Computer Society.

[4] H. Y. Chang, S. F. Wu, and Y. F. Jou, "Real-time protocol analysis for link state routing," submitted to Transaction on Information and System Security for review, 1999.

[5] S. F. Wu, Y. F. Jou, F. Wang, H. Chang, D. Qu, C. Sargor, F. Gong, and R. Cleaveland, "JiNao: design and implementation of a scalable intrusion detection system for the OSPF routing protocol," to appear in the Journal of Computer Networks and ISDN Systems.

[6] Y. F. Jou, F. Gong, C. Sargor, X. Wu, S. F. Wu, H. C. Chang, and F. Wang, "Design and implementation of a scalable intrusion detection system for the protection of network infrastructure," in *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 1999, pp. 422–434.

[7] F. Wang, F. Gong, F. Wu, and R. Narayan, "Intrusion detection for link state routing protocol through integrated network management," in *IEEE 8th International Conference on Computer Communication and Network (IC3N*, October 1999.

[8] F. Wang, "Vulnerability analysis and protection for link state routing protocol," `http://worf.mcnc.org/~fwang2/docs/pre99.ps`, September 1999.

[9] Teresa F. Lunt, "A survey of intrusion detection techniques," *Computers & Security*, vol. 4, no. 12, pp. 405–418, December 1993.

[10] F. Wang, *Vulnerability Analysis, Intrusion Prevention and Detection for Link State Routing Protocols*, Ph.D. thesis, North Carolina State University, Raleigh, NC, August 2000.

[11] K. Mani Chandy, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, Febuary 1985.

[12] K. Mani Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.

[13] J. Moy, "OSPF Version 2," Internet RFC 2178, July 1997.

[14] Feiyi Wang, Fengmin Gong, and Felix S. Wu, "A property oriented fault detection approach for link state routing protocol," in *Proceedings Ninth International Conference on Computer Communications and Networks*, Ton Engbersen and E.K. Park, Eds., Las Vegas, Nevada, October 16-18 2000, pp. 114–119.

[15] FNIISC Project, "Fault-Tolerant Networking Through Intrusion Identification and Secure Compartments," `http://fniisc.east.isi.edu/`, 2000.