IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 1, 2019, accepted February 17, 2019, date of publication March 5, 2019, date of current version April 1, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2903150

DAPV: Diagnosing Anomalies in MANETs Routing With Provenance and Verification

TENG LI[1], JIANFENG MA[1], QINGQI PEI[2], HOUBING SONG[3], (Senior Member, IEEE), YULONG SHEN[4], AND CONG SUN[1]

[1]School of Cyber Engineering, Xidian University, Xi'an 710071, China
[2]Shaanxi Key Laboratory of Blockchain and Security Computing, Xidian University, Xi'an 710071, China
[3]Department of Electrical and Computer Engineering, West Virginia University, Montgomery, WV 25801, USA
[4]School of Computer Science, Xidian University, Xi'an 710071, China

Corresponding author: Teng Li (litengxidian@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602357 and Grant 61872283, in part by the Shaanxi Science and Technology Coordination and Innovation Project under Grant 016TZC-G-6-3, in part by the China Postdoctoral Science Foundation Funded Project under Grant 2018M640962, in part by the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209, in part by the Natural Science Basis Research Plan in Shaanxi Province of China under Grant 2016JM6034, and in part by the China 111 Project under Grant B16037.

ABSTRACT Routing security plays an important role in the mobile ad hoc networks (MANETs). Despite many attempts to improve its security, the routing mechanism of MANETs remains vulnerable to attacks. Unlike most existing solutions that prevent the specific problems, our approach tends to detect the misbehavior and identify the anomalous nodes in MANETs automatically. The existing approaches offer support for detecting attacks or debugging in different routing phases, but many of them cannot answer the absence of an event. Besides, without considering the privacy of the nodes, these methods depend on the central control program or a third party to supervise the whole network. In this paper, we present a system called DAPV that can find single or collaborative malicious nodes and the paralyzed nodes which behave abnormally. DAPV can detect both direct and indirect attacks launched during the routing phase. To detect malicious or abnormal nodes, DAPV relies on two main techniques. First, the provenance tracking enables the hosts to deduce the expected log information of the peers with the known log entries. Second, the privacy-preserving verification uses Merkle Hash Tree to verify the logs without revealing any privacy of the nodes. We demonstrate the effectiveness of our approach by applying DAPV to three scenarios: 1) detecting injected malicious intermediated routers which commit active and passive attacks in MANETs; 2) resisting the collaborative black-hole attack of the AODV protocol, and; 3) detecting paralyzed routers in university campus networks. Our experimental results show that our approach can detect the malicious and paralyzed nodes, and the overhead of DAPV is moderate.

INDEX TERMS Wireless network security, anomalies detection, distributed verification.

I. INTRODUCTION

Mobile Ad hoc Networks (MANETs) [1] are continuously self-configuring, infrastructure-less networks made of a collection of mobile devices without any centralized management. In such networks, each node can play a role as a host or a router cooperating with other nodes [2]. As the primary goal of MANETs routing, secure routing means to establish a correct and efficient path by the distributed nodes themselves and can transmit the data fast and correctly. Besides the passive attack in which the intruders attempt to eavesdrop on the communication, many kinds of active attacks tend to damage the network by changing the data which is more common and harmful in MANETs. A typical case is that the single-node adversary or the coordinate adversaries will compromise the network by direct attack or indirect attack after deceiving the routing. In the direct attack, the adversaries can tamper or drop the packets to make the network paralyzed, such as the black hole attack [3] and worm hole attack [4]. As for the indirect attack, the adversaries deliberately modify their own logs without influencing the data forwarding to confuse the network debugging system [5],

The associate editor coordinating the review of this manuscript and approving it for publication was Marco Anisetti.

such as PeerReview [6] and NetReview [7] which depend on the nodes' log to detect faults in the network. The capability to find such malicious adversaries or the nodes attacked by these offending nodes is also essential to a diverse set of network management tasks such as performing network accountability [8], identifying malicious and misbehaving nodes [9], and enforcing trust management policies in distributed systems [10].

In order to find the abnormal nodes launching direct or indirect attack and explain why such misbehaviors are triggered in the network without disclosing the node's privacy, we present DAPV, an automatic fault detection system for the routing of MANETs using provenance reasoning to trace back the faults that have occurred in MANETs until we find the causes of the faults. In this approach, we first spot the influenced nodes without human inspectors and then find out which node originally triggers such faults by using provenance reasoning. The provenance of an observed event in our method is a chain of events forming a back trace to link this event to its original causes. This chain is called the provenance of the event [11], [12]. If a request log entry is detected to be false on the destination node, DAPV can then trace the request log back to the previous-hop node, where another false log entry may also be found. We can continue to track the fault to another node and then we may finally deduce that the false log entry was caused by specific malicious behavior of the tracked node, e.g. package tampering or truncating. The provenance of DAPV can help us explain why such an event is abnormal or why such a log did not occur.

The abnormal events in MANETs can be either specified as the occurrence of something bad, e.g. wrong log entry on a destination node, or as the absence of something good, e.g. missing log entry on some node. Both kinds of abnormal events can lead to the compromised network. Existing solutions, SNooPY [13] or NetSight [14], provide forensic capabilities by permitting inspectors to track down the misbehaving node for identifying the occurrence of malicious behaviors. But these approaches fail to report the absence of good events since we do not have a valid starting point for the back-tracing, e.g. we do not know where the missing packets should have come from. On the contrary, our approach can perform good reasoning for the provenance on both kinds of abnormal events.

In our approach, we will verify the data packets and log entries to infer the invalid states of nodes and get the facts on provenance. This kind of attack detection usually requires the participating nodes to disclose details of the routing policy or their private log data [15], [16]. But for privacy reasons, the hosts of MANETs are reluctant to disclose all of these data. Hence, the requirement on privacy preservation of the nodes inherently conflicts with the provenance reasoning, thus adding a new dimension to the tracking and detection mechanism of routing fault detection. Recent work, PeerReview [6] and Y! [17], detected faults without considering the issue of privacy. To deal with this contradiction, we use the Merkle Hash Tree (MHT) and digital signature algorithm to protect the privacy of the nodes. MHT can provide zero-knowledge fault detection without directly checking through all of the original data. Digital signature algorithm can be used by the message recipient to ascertain the originator's identity and ensure whether the message has been altered during the transmission phase. To illustrate the efficiency and correctness of our approach, we use DAPV to debug several realistic problems in three applications: direct and indirect attacks in MANETs; single and multiple black hole attacks of AODV; and different kinds of attacks in routers. In summary, this paper makes the following contributions:

1. A decentralized approach for automatically detecting both direct and indirect attacks launched in routing discovery and data forwarding phase in MANETs without introducing any third party.
2. The approach can not only detect the malicious nodes who launched the attacks in the path but also spot the paralyzed routers which were attacked by the adversaries that may not be in the route path.
3. We provide both positive and negative provenance as well as a concrete algorithm for tracking single and collaborative malicious adversaries in MANETs.
4. Our approach can preserve the privacy of the nodes during the provenance by using reasoning and Merkle Hash Tree instead of looking through the plain log entries.

II. OVERVIEW

A. THREAT SCENARIO

MANETs are self-organizing, infrastructure-less and have changeable topology. These features add the difficulty to embedding a security mechanism into routing protocols of MANETs. According to the these characteristics, we consider a scenario in which the nodes can be categorized into three roles: the sender, the receiver and the intermediated routers. Only the message sender is initially trusted and there might be more than one malicious node, either independently or collaboratively performing the attack along multiple routing paths. In the routing discovery phase, a malicious node has the ability of camouflaging as a destination or declaring to have a shortest/fastest route to the destination. Several malicious nodes connected with each other can also deceive the source node by declaring that they have a shortest/fastest route to the destination, and the destination is imitated by one of the malicious nodes. The preparations done by malicious nodes in the routing discovery phase can help in the following data forwarding phase to rewrite, discard the packets (direct attack) or temper their own log (indirect attack) against the expected security requirement. More specifically, we consider a multi-node black-hole attack in MANETs, as well as the attacks with the misbehaviors of rewriting, discarding packets (direct attack) or tempering routing logs (indirect attack) during their data forwarding phase. Besides, we also study the routers suffering the attacks which make them paralyzed in the routing path, such as the

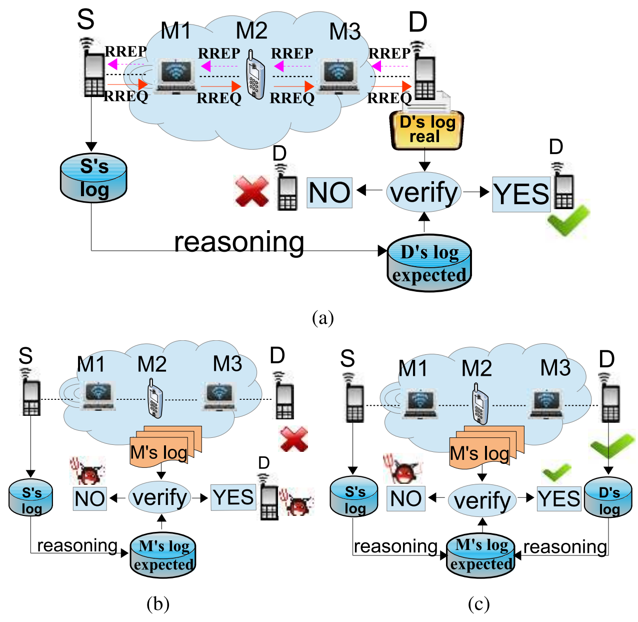


FIGURE 1. **The procedure of DAPV. (a) Initial misbehavior detection. (b) Direct attack detection. (c) Indirect attack detection.**

SSL attack [18], DOS attack [19], ARP spoofing attack [20], etc.

B. APPROACH

The overview of the architecture of DAPV is given in Figure 1. The detection of our approach is performed in the data forwarding phase. We assume to trust only the source node for our detection. This is different from the MOS (minimum observer sets) [21] which trusts both the source and destination to deduce the information of the medial routers. Our attacker model is more realistic to allow the malicious destination because according to the routing path established under a black-hole attack, we can neither ensure that the destination is a benign node, nor validate the information from the destination initially. We use the logs on the trusted source node to deduce the information that should be held by the destination node. Then, DAPV should verify whether the expected logs on the destination are consistent with its real log entries (see Figure 1(a)). If the expected log entry is absent in the destination, or the real log entries are verified to be false, DAPV uses positive or negative provenance to find the reason and the malicious node. After the provenance reasoning, we can distinguish whether the destination is malicious or it is just influenced by the malicious intermediated nodes (see Figure 1(b)). In the above two processes, we assume that the integrity of the destination and intermediated router's log should be held. It means that if they committed some malicious behavior, such as tempering or dropping some packets, they must log what they have done. In contrast, if the consistency between expected logs and real logs is verified, DAPV will then use the information from both the source and destination to verify whether the intermediate nodes behave well or commit attack by tempering their own log to confuse the network debugging system (see Figure 1(c)). In this process, we assume that only the integrity of the destination's log should be held. Finally, DAPV constructs the provenance graph to show the reasoning and verification link to the malicious node behavior.

With the consideration of privacy preservation during the provenance reasoning and verification, we built an MHT for each node consistent with its log entries. Instead of checking the plain log data of the nodes, DAPV chooses to verify the log entry from the leaf node of the MHT all the way to the root node. By comparing the calculated root hash value, DAPV can realize privacy preserving verification. To achieve the robustness of verification, DAPV leverages DSA to avoid the malicious intermediate nodes tempering the verification parameters.

III. DESCRIPTION OF DAPV

In this section, we present the details of our anomalies diagnosing mechanism with provenance and reasoning.

A. BASIC PROVENANCE RULES

Network Datalog (NDlog) is based on the Datalog [22]. In NDlog, the state of the node is described as a set of declarative rules. Each rule has the form $\mathtt{p}$:- $\mathtt{q}_1, \mathtt{q}_2, \mathtt{q}_3, \ldots, \mathtt{q}_n \ldots$, which means as "$\mathtt{q}_1$ and $\mathtt{q}_2$ and $\mathtt{q}_3$ and $\mathtt{q}_n$ implies $\mathtt{p}$". For instance, the rule `A(@X,S):- B(@X,P), S = 2*P` says that the left tuple `A(@X,S)` should be derived on node `X` if there exists a tuple `B(@X,P)`, and `S = 2*P`. Commas separating the predicates in the right side represent logical conjunctions (AND). The symbol @ specifies the node on which the tuple resides. Rules can also include tuples from different nodes and deduce the new rules in a recursive fashion. For instance, from the rule `A(@X,S):- B(@X,P)` and the rule `B(@X,P):- C(@Y,P)`, we can deduce a derived rule `A(@X,S):- C(@Y,P)`.

We propose the basic rules in NDlog for both routing discovery and data forwarding as shown in Table 1. The predicates used by these reasoning rules can be either real or expected. The real predicates are validated on the host according to its routing logs, and the expected predicates are derived by the subsequent credible reasoning on real predicates used later by the verification. For instance, the reasoning can derive the expected predicate, e.g. `A(@X,S)`, from the valid real predicate, e.g. `C(@Y,P)`, based on the rule `A(@X,S) :- C(@Y,P)`.

In Table 1, `C` and `R` represent the sender's and receiver's storage place respectively. `S` represents the sender, `D` means the receiver and `SEQ` is the corresponding sequence number. Intermediate routers' messages are stored in the place of `M` (for different intermediated routers it can be $\mathtt{M}_1, \mathtt{M}_2 \ldots$). `MSG` is the message that the source node wants to send to the destination host. The log is strongly relevant to the time, therefore we use a timestamp `T` of a time slot to provide historical queries. The duration of this time slot can vary to ensure the real predicates used in the reasoning are included in this slot.

TABLE 1. The deduction rules for DAPV.

1. `getRequest(@M,S,D,SEQ,T)`	`:-sendRequest(@C,S,D,SEQ,T)`
2. `reqForward(@M,S,D,SEQ,STAUS,T)`	`:-getRequest(@M,S,D,SEQ,T), STAUS='YES/NO'`
3. `getRequest(@R,S,D,SEQ,T)`	`:-reqForward(@M,S,D,SEQ,STAUS,T), STAUS='NO'`
4. `sendReply(@R,S,D,SEQ,T)`	`:-getRequest(@R,S,D,SEQ,T), routeAvailable(@R,S,D,SEQ,T)`
5. `getReply(@C,S,D,SEQ,STAUS,T)`	`:-repForward(@M,S,D,SEQ,STAUS,T), sendReply(@R,S,D,SEQ,T), STAUS='YES'`
6. `findDest(@C,S,D,SEQ,STAUS,T)`	`:-getReply(@C,S,D,SEQ,STAUS,T), getRequest(@R,S,D,SEQ,T)`
7. `reachDest(@M,S,D,SEQ,STAUS,T)`	`:-findDest(@C,S,D,SEQ,STAUS,T)`
8. `dataLink(@C,S,D,SEQ,STAUS,T)`	`:-findDest(@C,S,D,SEQ,STAUS,T), sendRequest(@C,S,D,SEQ,T), STAUS='YES/NO'`
9. `authSend(@R,S,D,SEQ,MSG,T)`	`:-dataLink(@C,S,D,SEQ,STAUS,T), msgRequest(@C,S,D,SEQ,MSG,T)`
10. `msgForward(@M,S,D,SEQ,MSG,T)`	`:-authSend(@R,S,D,SEQ,MSG,T), sendMsg(@C,S,D,SEQ,MSG,T)`
11. `msgRecv(@R,S,D,SEQ,MSG,T)`	`:-msgForward(@M,S,D,SEQ,MSG,T)`
12. `msgRecvAck(@C,S,D,SEQ,MSG,T)`	`:-msgRecv(@R,S,D,SEQ,MSG,T), msgRecvReplyForward(@M,S,D,SEQ,MSG,T)`
13. `reSendMsg(@C,S,D,SEQ,MSG,T)`	`:-recvError(@C,S,D,ERROR,T), authSend(@R,S,D,SEQ,MSG,T)`

Rules 1-8 are for the routing discovery phase. As a sender wants to find a route to the destination, it first broadcasts a routing request to its neighbors, and a real predicate `sendRequest(@C,S,D,SEQ,T)` is logged. The predicate means `S` wants to find a routing path to `D` with a destination sequence number `SEQ` and this request is logged at place `C`. When a neighbor `M` gains the request, it first looks for the routing table, if it does not have the route to the destination, the neighbor continues to broadcast the routing request to other nodes (a predicate `reqForward(@M,S,D,SEQ,STAUS,T)` is confirmed to be real from Rule 2 and the real `getRequest(@M,S,D,SEQ,T)`). When each time the intermediate node can correctly forward the destination query, the destination is expected to log a predicate `getRequest(@R,S,D,SEQ,T)`, (Rule 3). When the destination gets the request and checks that the route is available in the same time slot, it will send the reply to the source node (Rule 4). If the intermediated routers also credibly forward the reply, the source node will get the reply message (Rule 5). When the recipient gets the request and the source node gets the reply, we can locate the destination by confirming the expected `findDest(@C,S,D,SEQ,STAUS,T)` to be real (Rule 6). When the source node has the route to the destination, the intermediated nodes will have the link to the destination as well (Rule 7). Once the sender finds the destination, the data link is established (Rule 8).

Rule 9-13 are for the data forwarding phase. When the source node proposes the message sending request `msgRequest(@C,S,D,SEQ,MSG,T)`, the benign destination should log an authorization record to the request (Rule 9). After the authorization (the predicate `authSend(@R,S, D,SEQ,MSG,T)` becomes real), the source node begins to send the message and the intermediate routers should keep the promise to forward the message (Rule 10). If the intermediate routers behave well, the recipient node will get the message (Rule 11) and send the acknowledgement message to the source node (Rule 12). If the sender has been authorized to send the message but receives an error, it will resend the message again (Rule 13).

Except for the primary rules presented in Table 1, there are auxiliary NDlog rules to support the correct running of our whole reasoning process. For example, some definition of predicates may allow a necessary prerequisite assumption that if the left-sided expected predicate τ can only be deduced by the rule τ :- α, β, then the validation of τ implies the right-sided predicates α and β must be real. That means we also have α :- τ and β :- τ. This necessary prerequisite assumption can hold on at least Rule 3, 5, 6, 8, 9, 12 and 13.

B. PROVENANCE VERIFICATION

During provenance reasoning, DAPV should verify the real router log on its consistency with the expected log. When the expected log predicate and the real log belong to different parties, the verification requires the owner of the real log to release its log data [6], which violates the requirement on privacy preservation of the individual node. In order to conduct a privacy-preserving verification, we introduce the Merkle Hash Tree to achieve the goal of privacy protection and use a digital signature algorithm to assist the verification phase.

1) MERKLE HASH TREE

Merkle Hash Tree (MHT) is a kind of binary hash tree, each of whose non-leaf node is labeled with the hash of the labels or values of its children nodes, and every leaf node is labeled with the hash value of real data [23]. MHT is commonly used in efficient and secure verification of the contents of large data structures. We designed a kind of MHT that is used for constructing the node's log for privacy-preserving verification. As a kind of binary tree, the edges of MHT from each parent to its two children are tagged with 0 and 1 respectively (see Figure 2). The purpose of verification is to ensure the expected actions or messages derived in the reasoning phase are satisfied by the real log of related nodes. The process consists of three steps: predicate encoding, MHT building, and verification. Before building the tree, we should first encode the information. Take the following predicate encoding as an example. `reqForward(@M,` S_1, D_1, `SEQ,STAUS'NO',T)` can be encoded by specifying S_1 = "01", D_1 = "0110", SEQ = "010, $STAUS(NO)$ = "0

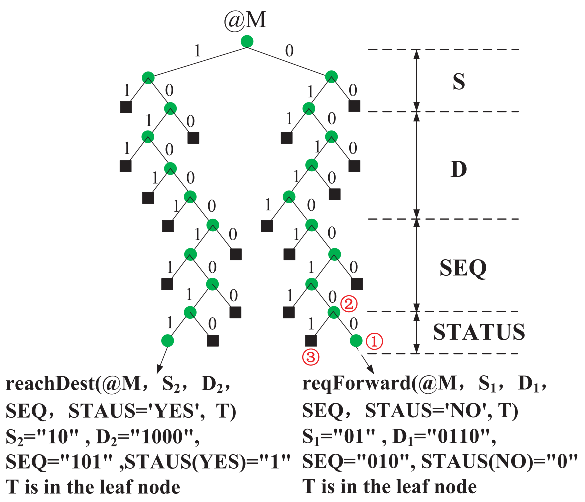


FIGURE 2. Merkle hash tree.

involved in the tree. The predicate $\texttt{reachDest}(@M, S_2, D_2, \texttt{SEQ}, \texttt{STAUS='YES'}, \texttt{T})$ can be encoded by specifying S_2 = "10", D_2 = "1000", SEQ = "101", STAUS(YES)="1". The value of T is in the leaf node. How many bits the variables, such as S_1 and S_2, cost for the predicate encoding depend on the count of nodes and variables. Our goal is to distinguish each predicate on different nodes.

After $\texttt{reqForward}(@M, S_1, D_1, \texttt{SEQ}, \texttt{STAUS='NO'}, \texttt{T})$ is encoded as a string, e.g. "0101100100", we use it to build the tree. Each node will search its real log to find out the predicating log, which is used to construct the MHT. We label the node's left child as 1 and right child as 0. Then, we can calculate the hash value of each node i: $H_i = H(\texttt{bit\_data}_{(i)} \parallel \texttt{parent\_bit\_data}_{(i)}) \parallel H_{\text{left\_child(i)}} \parallel H_{\text{right\_child(i)}})$. $\texttt{bit\_data}_{(i)}$ and $\texttt{parent\_bit\_data}_{(i)}$ are the value of current node and parent node respectively, which can be either 0 or 1. $H_{\texttt{left\_child(i)}}$ and $H_{\texttt{right\_child(i)}}$ are the hash value of the node(i)'s left child and right child respectively. They will be empty if the current node is the leaf node. We can calculate the hash value from the leaf node to the root, and this value will be published. That means every node may know the root hash value of any other nodes.

In the verification phase, the router S_1 infers the log entries $\texttt{reqForward}(@M, S_1, D_1, \texttt{SEQ}, \texttt{STAUS} = \texttt{'NO'}, \texttt{T})$ of D_1. Then, S_1 will require in advance the root hash value on all of the log information owned by D_1. Next, S_1 will ask D_1 whether there exits the inferred log. To answer the question, D_1 starts from the MHT root and follows the edges (in Figure 2, the right green branch) using the string "0101100100" to find the leaf node (in Figure 2, node 1) in its own MHT and then provides bit_data, parent_bit_data ($H_{\text{left\_child}}$ and $H_{\text{right\_child}}$ are empty now) of the leaf for S_1. Then, S_1 can use these parameters to calculate the hash value of the leaf node using the mentioned function. Next, D_1 provides the parameters of the leaf node's parent node (in Figure 2, node 2), bit_data, parent_bit_data, $H_{\texttt{left\_child}}$ and $H_{\texttt{right\_child}}$. bit_data = 0, parent_bit_data = 1, $H_{\texttt{left\_child}}$ is the hash value of node 3 and $H_{\texttt{right\_child}}$ is the hash value of node 1. Then, S_1 can use these parameters to calculate the hash of node 2. In this way, S_1 can calculate the MHT's hash from the leaf node all the way to the root node. If the calculated hash equals the former published value, the verification result is true. Due to the decentralization of our method, the hosts in DAPV work in a coordinated way and will exchange logs or parameters. To avoid tempering of these parameters, we use the Digital Signature Algorithm (DSA) to prove to the recipient that the messages or the parameters are signed by the originator.

By using MHT, the verification is done on the verifier and the prover is responsible for providing the parameters during the calculating. Instead of receiving the plain log entries, the verifier requires the parameters, which are bits data and hash values, and cannot know the other privacy of the prover. Because the verifier ask for the root hash value of the prover before calculating, the prover cannot forget that it has such a message in its MHT in advance for it does not know the verified message. In this way, we can achieve the goal of zero-knowledge verification but preserve the privacy of the nodes.

C. PROVENANCE REASONING

To detect the mentioned direct and indirect attack, we should first do the initial detection towards the destination. During the initial detection, we use the log entry of the trusted sender to deduce the expected log entry in the destination. After verifying the real log with the expected log on the destination, we can know which kind of attack the MANETs are suffering. If the log on the destination is not correct, DAPV goes to the direct attack detection. We assume that the integrity of the destination and intermediated router's log should be held during the initial attack detection and the direct attack detection. If the log on the destination is correct, DAPV goes to the indirect attack detection to check if the intermediated routers have changed their own log. During this phase, we assume that only the integrity of the destination's log should be held.

There are both false log or absent log events in the routing phase. We categorize the events into positive and negative events. For the positive event, such as the false log entry or just the normal log entry, we can use the positive provenance to derive a back trace recursively until we find the original attacker. The goal of positive provenance is to detect inconsistent or incorrect states in the faulty component [24]. Negative provenance is used to explain why such an event is missing or absent - e.g., why such a routing request did not arrive on the destination or why the acknowledgement message did not appear on the source node. Although these events cannot be directly explained by positive provenance, we can also form a similar back trace that explains where the missing log or packets should have come according to the basic rules that we mentioned in Table 1. The goal of negative provenance is to find out the ways in which a missing event should have occurred and show the reason why it did not come to pass.

1) INITIAL MISBEHAVIOR DETECTION

We use the DAPV to supervise the initial symptom of an attack. In MANETs routing, from the beginning, we focus on the correctness of the destination. Thus, we should first know what information should have arrived at the recipient and then we can conduct the verification. With the above basic rules in Sec.3.1, we can deduce the expected destination information. We call this process as C→R (source node deduces the destination node).

$$\texttt{sendRequest(@C,S,D,SEQ,T)} \rightarrow \texttt{getRequest(@R,S,D,SEQ,T)} \quad (1)$$

$$\texttt{getReply(@C,S,D,SEQ,T)} \rightarrow \texttt{sendReply(@R,S,D,SEQ,T)} \quad (2)$$

$$\texttt{findDest(@C,S,D,SEQ,T)} \wedge \texttt{getReply(@C,S,D,SEQ,T)} \rightarrow \texttt{getRequest(@R,S,D,SEQ,T)} \quad (3)$$

$$\texttt{dataLink(@C,S,D,SEQ,STAUS,T)} \wedge \texttt{msgRequest(@C,S,D,SEQ,MSG,T)} \rightarrow \texttt{authSend(@R,S,D,SEQ,MSG,T)} \quad (4)$$

$$\texttt{msgRecvAck(@C,S,D,SEQ,MSG,T)} \rightarrow \texttt{msgRecv(@R,S,D,SEQ,MSG,T)} \quad (5)$$

$$\texttt{reSendMsg(@C,S,D,SEQ,MSG,T)} \wedge \texttt{recvError(@C,S,D,ERROR,T)} \rightarrow \texttt{authSend(@R,S,D,SEQ,MSG,T)} \quad (6)$$

We start reasoning from the source node, which is the only trusted node from the beginning. We use the sender's log stored at the place of C to infer the logs expected to be stored at R, e.g. the log entry getReply(@C,S,D,SEQ,T) is on the source node. According to Rule 2, the expected predicates of the receiver can be deduced (sendReply(@R,S,D,SEQ,T)), which should comply with the real log on the destination. Then the source node uses MHT to verify the deduced predicate with the real log entries on the destination. If the result is false, we know that there may exist intermediate malicious nodes that influence the message transmission. Hence, we go to the direct attack detection. If the result is correct, we should also check the log of the intermediated routers to find if there is any indirect attack.

2) DIRECT ATTACK DETECTION

In both the routing discovery phase and data forwarding phase, the adversary can launch a direct attack when the intermediate routers may deliberately temper or discard the forwarded messages to the receiver. The obvious phenomenon of this kind of attacks is the false log on the destination. If the initial misbehavior detection reports a result of a false log, we know that there must be at least one malicious router which has tempered or discarded the forwarded messages, and these malicious nodes can either be the intermediate nodes or the destination itself. A concrete case for such an attack is the black hole attack. We should use the correct log in the source node to deduce the expected log of the intermediate routers. We call this process as C→M (source node deduces intermediate routers).

$$\texttt{sendRequest(@C,S,D,SEQ,T)} \rightarrow \texttt{getRequest(@M,S,D,SEQ,T)} \quad (7)$$

$$\texttt{findDest(@C,S,D,SEQ,T)} \wedge \texttt{getReply(@C,S,D,SEQ,T)} \rightarrow \texttt{reqForward(@M,S,D,SEQ,STAUS,T)} \quad (8)$$

$$\texttt{getReply(@C,S,D,SEQ,STAUS,T)} \rightarrow \texttt{repForward(@M,S,D,SEQ,STAUS,T)} \quad (9)$$

$$\texttt{findDest(@C,S,D,SEQ,STAUS,T)} \rightarrow \texttt{reachDest(@M,S,D,SEQ,STAUS,T)} \quad (10)$$

$$\texttt{dataLink(@C,S,D,SEQ,'Yes',T)} \wedge \texttt{msgRequest(@C,S,D,SEQ,MSG,T)} \wedge \texttt{sendMsg(@C,S,D,SEQ,MSG,T)} \rightarrow \texttt{msgForward(@M,S,D,SEQ,MSG,T)} \quad (11)$$

$$\texttt{msgRecvAck(@C,S,D,SEQ,MSG,T)} \rightarrow \texttt{msgRecvReplyForward(@M,S,D,SEQ,MSG,T)} \quad (12)$$

In most cases, the route path from the sender to the receiver has more than one hop, such as N_1-M_3-M_6-M_7-N_9, and the intermediate routers are M_3, M_6 and M_7. DAPV checks them in a reverse order until we find the first benign node whose nearest successor's log is false. For instance, if N_9 is detected to be false and we have deduced findDest(@C,S,D,SEQ,T) and getReply(@C,S,D,SEQ,T) from the log entries of sender S, then with the deducing rule (8), the sender will know that reqForward(@M,S,D,SEQ,STAUS,T) should exist on the intermediated routers. After that, the sender uses MHT to verify the expected log with the real logs on M_7, M_6 and M_3 respectively. If M_7 and M_6 are incorrect and M_3 is correct, we confirm that M_3 is the malicious node which affected the following nodes on the routing path. The order of the candidate node is critical to our detection. The reason why we check the nodes in a reverse order is that the nodes closest to the fault node are more likely to record false logs than the far away nodes, and detecting the false log is much faster than checking the correct log.

3) INDIRECT ATTACK DETECTION

It is difficult to detect the indirect attack in MANETs routing because the attacker does not influence the transmitted packets: it eavesdrops on the messages or tempers its own data log to prevent the network administrator from sniffing out the faults. Encryption is the countermeasure to eavesdropping which is not discussed in this paper. Instead, we focus on the router's log tempering in this paper.

If the result of initial misbehavior detection is correct, we can also check whether there exist indirect attacks. We can use both source node's log and the verified destination's log to infer the expected predicates of the intermediate log. We call this process as C+R→M (source node and destination node deduce intermediate routers).

```
getReply(@C,S,D,SEQ,T)
    ∧ sendReply(@R,S,D,SEQ,T)                (13)
    → repForward(@M,S,D,SEQ,STAUS,T)
sendMsg(@C,S,D,SEQ,MSG,T)                    (14)
    ∧ authSend(@R,S,D,SEQ,MSG,T)
    → msgForward(@M,S,D,SEQ,MSG,T)
msgRecvAck(@C,S,D,SEQ,MSG,T)
    ∧ msgRecv(@R,S,D,SEQ,MSG,T)
    → msgRecvReplyForward                    (15)
      (@M,S,D,SEQ,MSG,T)
authSend(@R,S,D,SEQ,MSG,T)
    ∧ msgRequest(@C,S,D,SEQ,MSG,T)
    → reachDest(@M,S,D,SEQ,T)                (16)
```

As we get the expected log of the intermediate routers, the routers provide the parameters of the MHT to the sender combining it with DSA to guarantee the security of the transmission. After the verification, we can know whether MANETs routing suffers from the indirect attack.

D. PROVENANCE GRAPH

Provenance can be represented as a Directed Acyclic Graph (DAG) in which the vertices are events and the edges show direct causal relationships. We try to construct a provenance graph consisting of multiple event chains which can reach the attack nodes. First, we define some types of event vertices:

- EXIST($[t_1, t_2]$, N, τ): Tuple τ existed on node N from time t_1 to t_2;
- ABSENT($[t_1, t_2]$, N, τ): Tuple τ was absent on node N in time interval $[t_1, t_2]$;
- DERIVE($[t_1, t_2]$, D, τ, τ_D, r): Tuple τ on node N was derived by the tuple set τ_D via rule r;
- CORRESPOND($[t_1, t_2]$, N, τ): Tuple τ was correspond with the expected tuple on node N in $[t_1, t_2]$;
- NCORRESPOND($[t_1, t_2]$, N, τ): Tuple τ was not correspond with the expected tuple on node N in $[t_1, t_2]$;
- TEMPER($[t_1, t_2]$, N, τ): Tuple τ was tempered by node N in $[t_1, t_2]$;
- DISCARD($[t_1, t_2]$, N, τ): Tuple τ was discarded by node N in $[t_1, t_2]$;
- DIRECT_ATTACK($[t_1, t_2]$, N, τ): Node N launched direct attack and tuple τ was not correspond with the expected one;
- INDIRECT_ATTACK($[t_1, t_2]$, N, τ): Node N launched indirect attack and tuple τ was discarded or tempered by intermediated routers;

The edges between the vertices include the relationship of the tuples which have been denoted by the basic rules.

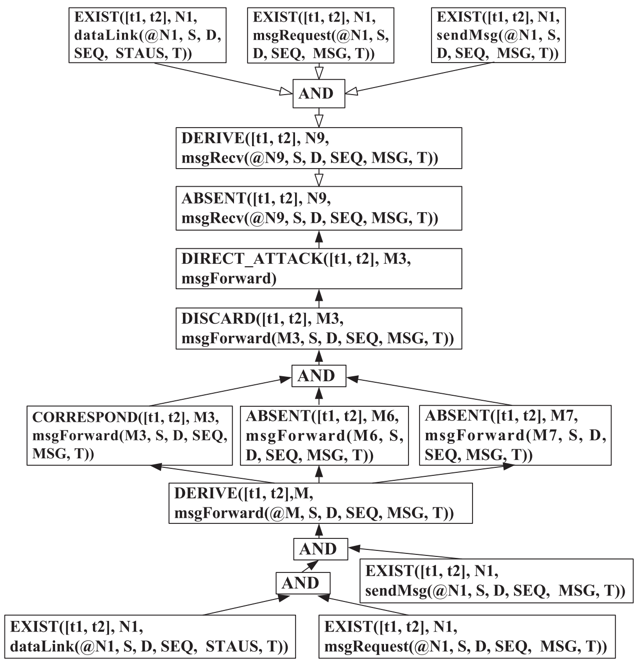

EXIST([t1, t2], N1, dataLink(@N1, S, D, SEQ, STAUS, T))
EXIST([t1, t2], N1, msgRequest(@N1, S, D, SEQ, MSG, T))
EXIST([t1, t2], N1, sendMsg(@N1, S, D, SEQ, MSG, T))
AND
DERIVE([t1, t2], N9, msgRecv(@N9, S, D, SEQ, MSG, T))
ABSENT([t1, t2], N9, msgRecv(@N9, S, D, SEQ, MSG, T))
DIRECT_ATTACK([t1, t2], M3, msgForward)
DISCARD([t1, t2], M3, msgForward(M3, S, D, SEQ, MSG, T))
AND
CORRESPOND([t1, t2], M3, msgForward(M3, S, D, SEQ, MSG, T))
ABSENT([t1, t2], M6, msgForward(M6, S, D, SEQ, MSG, T))
ABSENT([t1, t2], M7, msgForward(M7, S, D, SEQ, MSG, T))
DERIVE([t1, t2],M, msgForward(@M, S, D, SEQ, MSG, T))
AND
AND
EXIST([t1, t2], N1, sendMsg(@N1, S, D, SEQ, MSG, T))
EXIST([t1, t2], N1, dataLink(@N1, S, D, SEQ, STAUS, T))
EXIST([t1, t2], N1, msgRequest(@N1, S, D, SEQ, MSG, T))

FIGURE 3. Positive provenance, as returned by DAPV.

Tuples can exist on a node because they are derived by other tuples. With MHT, we can know whether the tuples are absent, correspond or do not correspond with the expected tuples. At last, we can find which malicious node tempered or discarded the messages. There are many other logs, such as the log entries of device booting, that we cannot use for deducing according to our basic rules, and we should not involve these irrelevant logs into our log set. We first prone these irrelevant logs and choose the relevant logs according to those in the basic rules in Table 1. When the new round discovery phase starts, the sequence number of the request message sent from the source node increases from 1. So we put all of the logs according to the basic rules into the log set during the time intervals before the send command starts a new round routing discovery phase.

1) GRAPH CONSTRUCTION ALGORITHM

Provenance systems like ExSPAN [25] rely on a materialized provenance graph: while the distributed system is executing, they build some representation of the vertices and edges in the graph, and they respond to queries by projecting out the relevant subtree. This approach does not suit for the negative provenance because the provenance graph is typically infinite [17].

For the above problem, DAPV provides the provenance graph according to our attack detection phases. DAPV only makes the nodes as well as their relevant log of the deducing phase involved in the provenance graph. The positive provenance graph of DAPV constructs two parts: the initial attack derive part (Figure 3, the event vertices with the hollow arrow) and the attack provenance part (Figure 3, the event

Algorithm 1 Provenance Graph Construction Algorithm

Input: *intermediated_route_list* $(m_1, m_2, m_3 \ldots m_n)$, containing intermediated routers; τ_C, source's log entries; τ_D, destination's log entries.

Output: Use *Graph_Draw*() function to draw the provenance graph.

```
$\tau_S = \text{EXIST}([t_1, t_2], S, \tau)$;
$\tau'_D = \text{DERIVE}([t_1, t_2], D, \tau, \tau_S, r)$;
Graph_Draw$(\tau_S, r)$;
intermediated_route_list$(m_1, m_2, m_3 \ldots m_n)$;
$result = \text{MHT\_Verify}(D, \tau'_D)$;
if $result == \text{NCORRESPOND}([t_1, t_2], D, D_\tau)$ then
    Graph_Draw$(result, initial\_derive)$;
    $\tau_M = \text{DERIVE}([t_1, t_2], M, \tau, \tau_S, r)$;
    Graph_Draw$(\tau_M, r)$;
    for $i = n$ to 1 do
        $result = \text{MHT\_Verify}(M_i, \tau_M)$;
        if $result == \text{ABSENT}([t_1, t_2], M_i, \tau_M)$ then
            Graph_Draw$(result, provenance\_derive)$;
            CONTINUE;
        else
            BREAK;
        end if
    end for
    if i== n then
        $result = \text{DISCARD}([t_1, t_2], D, \tau_M)$;
        Graph_Draw$(result, provenance\_derive)$;
        $result = \text{DIRECT\_ATTACK}([t_1, t_2], D, \tau_M)$;
        Graph_Draw$(result, provenance\_derive)$;
        RETURN;
    else
        $result = \text{DISCARD}([t_1, t_2], M_i, \tau_M)$;
        Graph_Draw$(result, provenance\_derive)$;
        $result = \text{DIRECT\_ATTACK}([t_1, t_2], M_i, \tau_M)$;
        Graph_Draw$(result, provenance\_derive)$;
        RETURN;
    end if
else
    $\tau_M = \text{DERIVE}([t_1, t_2], M, \tau, \tau_S + \tau_D, r)$;
    Graph_Draw$(\tau_M, r)$;
    for $i = 1$ to $n$ do
        $result = \text{MHT\_Verify}(M_i, \tau_M)$;
        if $result == \text{ABSENT}([t_1, t_2], M_i, \tau_M) \parallel$
        $result == \text{NCORRESPOND}([t_1, t_2], M_i, \tau_M)$ then
            Graph_Draw$(result, provenance\_derive)$;
            $result == \text{TEMPER}([t_1, t_2], M_i, \tau_M)$;
            Graph_Draw$(result, provenance\_derive)$;
            $result = \text{INDIRECT\_ATTACK}([t_1, t_2], M_i, \tau_M)$;
            Graph_Draw$(result, provenance\_derive)$;
        else
            Graph_Draw$(result, provenance\_derive)$;
            CONTINUE;
        end if
    end for
    RETURN;
end if
```

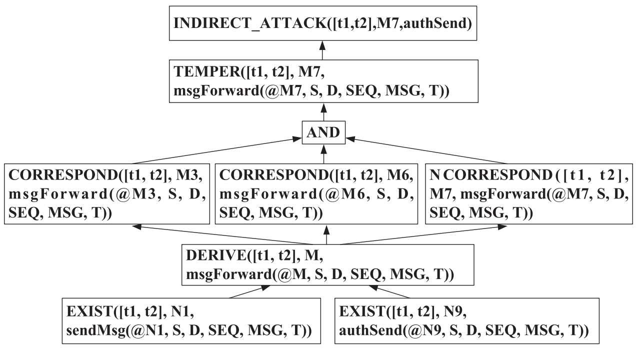

INDIRECT_ATTACK([t1,t2],M7,authSend)
TEMPER([t1, t2], M7, msgForward(@M7, S, D, SEQ, MSG, T))
AND
CORRESPOND([t1, t2], M3, msgForward(@M3, S, D, SEQ, MSG, T))
CORRESPOND([t1, t2], M6, msgForward(@M6, S, D, SEQ, MSG, T))
NCORRESPOND([t1, t2], M7, msgForward(@M7, S, D, SEQ, MSG, T))
DERIVE([t1, t2], M, msgForward(@M, S, D, SEQ, MSG, T))
EXIST([t1, t2], N1, sendMsg(@N1, S, D, SEQ, MSG, T))
EXIST([t1, t2], N9, authSend(@N9, S, D, SEQ, MSG, T))

FIGURE 4. Negative provenance, as returned by DAPV.

vertices with the solid arrow), while the negative provenance graph of DAPV only has the attack provenance part (Figure 4, the event vertices with the solid arrow). The initial attack derive part can deduce the initial problems automatically. We also do the initial attack derive phase in negative provenance, but we do not construct it in the graph because the log on the destination is correct.

In Algorithm 1, τ_S is the event vertice set in the source node S using for the deducing (as shown in Figure 3, `datalink`, `msgRequest` and `sendMsg`). τ'_D is the deduced result set on destination node D. We show this deducing process in graph with Graph_Draw(τ_S, r). Then, we use the MHT to verify the deduced results with the real existing log on destination. If the verified result is false, we check the intermediated routers. First, we should also deduce the expected event vertice set τ_M of intermediated routers. Then, we use MHT to verify the routers in a reverse sequence. If all the intermediated routers behave well, we can proof that the destination itself is the malicious node. Otherwise, we can find the adversary launching direct attack. If all the verifications on the destination node are correct, we will go to the negative provenance, we use the tuples on both source node and destination node to deduce the intermediated routers' event vertice set τ_M and go to the verification phase like in the positive provenance. Algorithm 1 shows all the details of our graph construction.

Figure 3 is the answer to positive provenance in our system, we only return the misbehavior nodes derivation and verification in the provenance graph. DAPV first uses the log on the sender to deduce the log in the destination and verifies its existence by using MHT. Once the it finds the suspicious result, it starts to inquiries the intermediated routers and constructs the whole provenance graph. Finally, it finds out that the forwarding router R3 launched the active attack.

Figure 4 is the answer returned to negative provenance. The indirect attack detection consumes more time than the direct attack detection as DAPV will check all the correct logs in the destination. First in the destination verification phase, cause the indirect attack does not influence the log on the recipient, we will do all the check in `C`$\rightarrow$`R` without showing in the negative provenance graph. Then, we go to the C+R→M, we should also verify every intermediate routers because each of them may launch attacks by tempering or discarding its own log. Finally, it finds out that the forwarding router R7 launched the passive attack.

IV. EVALUATION

In this section, we evaluate DAPV in three scenarios: 1) resisting the collaborative black-hole attack of the AODV protocol; 2) detecting injected malicious intermediated routers which commit active and passive attacks in MANETs; 3) detecting the paralyzed routers in a university networks. To provide a baseline for comparisons, we aligned our experiments with our former work CRVad [26] and the original AODV protocol performance. CRVad is a baseline for DAPV because it can also verify the correctness of the nodes after the route was built. However, during the data forwarding phase DAPV adds the DSA mechanism and uses the simplest way to handle the multiple derivation as well as provide the positive and negative provenance to find out the malicious nodes.

A. EXPERIMENT SETUP

We used NS-3 [27] to do our first two experiments and also deployed DAPV in a university education network. We injected several faults into the MANETs during the routing discovery and data forwarding phase as shown in Table 2. Then, DAPV could detect the faults and find out the malicious nodes. We also mounted both a single-node and collaborative black hole attack over the AODV routing protocol. DAPV could distinguish the suspicious log entries with the expected logs and find out the adversaries during the routing discovery phase. Besides, we launched different attacks towards the network containing Cisco, Huawei and TP-Link routers. Then, we used DAPV to detect the anomalies in the network.

We conducted the reasoning process using an open source tool IRIS [28], and have proved that the conclusions we inferred are correct. All of the reasoning results we used have been written in NDlog and verified on IRIS. We built up multi-hop topology among different nodes. We used the Logging Module (NS_LOG_DEBUG) of NS-3 and syslogs in the routers for log sending and receiving messages. Then, each node can use this kind of information to build up its own Merkle Hash Tree. We use C++ to build the tree and conduct the root hash value verification phase with DSA in openssl.

DAPV can detect the misbehavior and identify the malicious nodes automatically. First, DAPV extracts the logs from the nodes with the logging module. Second, on the source node, DAPV uses its logs with the corresponding rules to deduce the expected logs on the target nodes (with the open source tool IRIS). Third, combined with MHT, the source node conducts the provenance verification to confirm whether the real log entries match the expected ones. Finally, according to our algorithms, DAPV returns the provenance graph for the whole deducing and verification phase which shows the exact malicious nodes and their attacking behaviors.

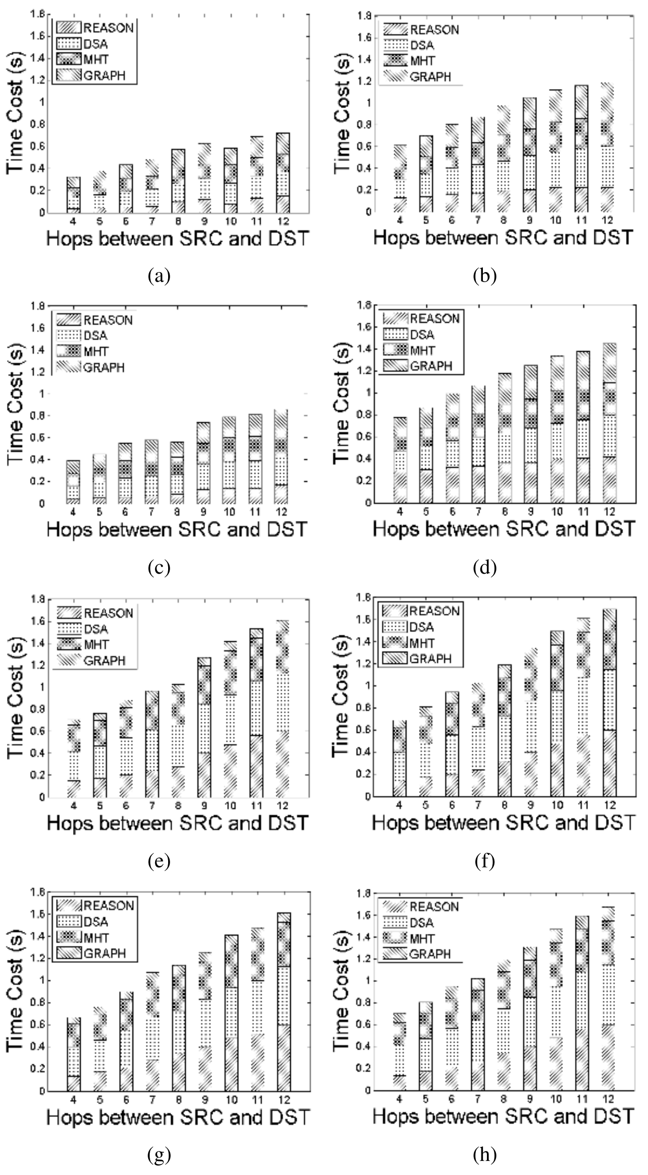


FIGURE 5. Turnaround time for the detection of the injected faults. (a) F1. (b) F2. (c) F3. (d) F4. (e) F5. (f) F6. (g) F7. (h) F8.

B. USABILITY: INJECTED FAULTS DETECTION

1) PERFORMANCE OF DETECTING THE INJECTED FAULTS

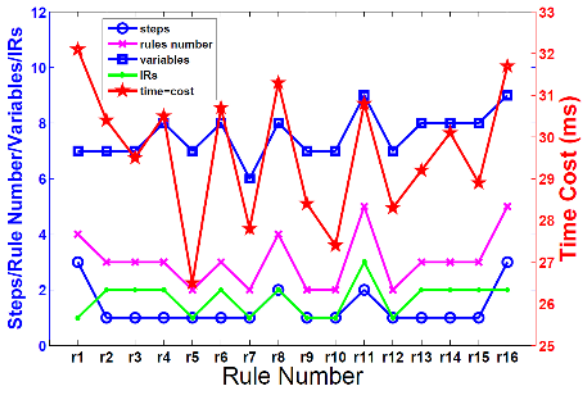
In our experiment, we inject eight different faults into our routing phases. F1-F4 are direct attacks that vary in the routing phase, malicious behavior and attacker number. These variants also exist in F5-F8 which are indirect attacks. We set the hops between the source node to the destination node from 4 hops to 12 hops, and we randomly generate the adversary's node. Among the direct attacks, we can see that the detection of the double malicious nodes consumes more time than the single malicious node detection. In Figure 5(b) and 5(d), we can see that the double malicious node detection needs more reasoning steps, MHT verification, DSA time consumption and nodes drawing. During our deducing and verification, we first check the logs of the routing discovery phase and then the data forwarding phase. The detection of

TABLE 2. Injected faults in our experiments.

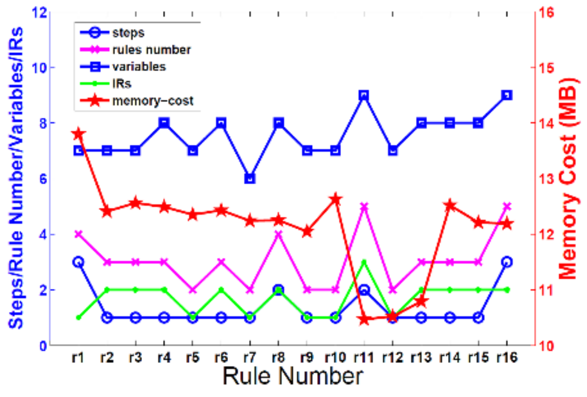
Faults	Attack	Routing Phase	Malicious Behavior	Attacker	influenced packets	Number of Attacks
F1	Direct	Discovery	Discard packets	Single	21/2321	42
F2	Direct	Discovery	Discard packets	Double	47/2568	76
F3	Direct	Forward	Temper packets	Single	73/3104	116
F4	Direct	Forward	Temper packets	Double	126/3678	168
F5	Indirect	Discovery	Discard log	Single	113/2896	125
F6	Indirect	Discovery	Discard log	Double	139/3139	168
F7	Indirect	Forward	Temper log	Single	151/3034	143
F8	Indirect	Forward	Temper log	Double	176/3679	167

the direct attack in the data forwarding phase costs more time than in the routing discovery phase, which is also proven in Figure 5(c) and Figure 5(a). Only when we confirm that all of the logs on the destination are correct, can we go to the indirect attack detection. This means that we will spend more time on the initial attack detection during the indirect faults detection than the direct faults detection which can be seen in Figure 5. As we have mentioned in Section III-D, we do not draw all of the correct initial detection during the indirect faults detection, so it will reduce the graphing time which can also be seen in Figure 5(e) - Figure 5(h). As Algorithm 1 shows, we do the indirect attack detection in each of the intermediated routers, so the time consumption of the single attacker and the double attacker in the indirect attack detection are approaching.

We use IRIS [28] to conduct our reasoning process. First, we use the NDlog language to write the basic transmitting rules. Then we extract the sending and receiving evidences from the log files of the NS3. The evidences which should exist on the nodes are reasoned before the verification and we just confirm these evidences to judge which node committed the attacks. The time and spatial cost of the deduction of each rule are given in Figure 6. We have observed four main factors, which are reasoning steps, rule numbers, variables and IRs, affecting the time and memory cost of DAPV. From the Figure 6, we can see the results of the time and memory cost varies with the changing of these factors. As for the time cost, we can see the strong relevance of the time cost with reasoning steps, rule numbers and variables. The IRs has little influences on the time costs. In memory cost, we can see the reasoning steps is the influential factor and the memory costs are closed with each other. The whole time and spatial cost is acceptable according to our experimental results.



(a)



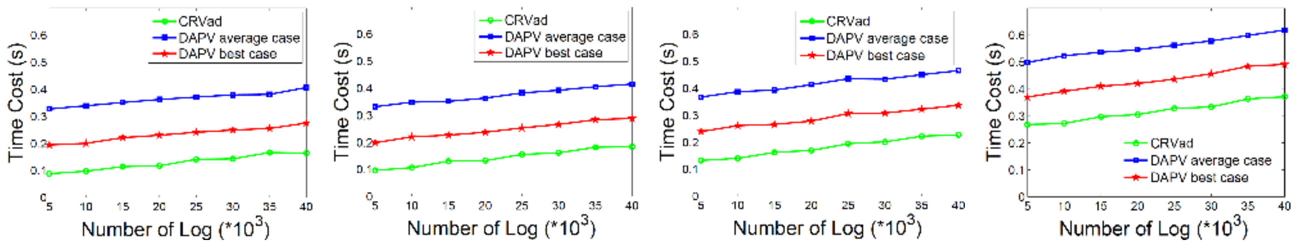
(b)

FIGURE 6. (a) Time and (b) memory cost of DAPV reasoning.

2) PROVENANCE VERIFICATION PERFORMANCE COMPARISON

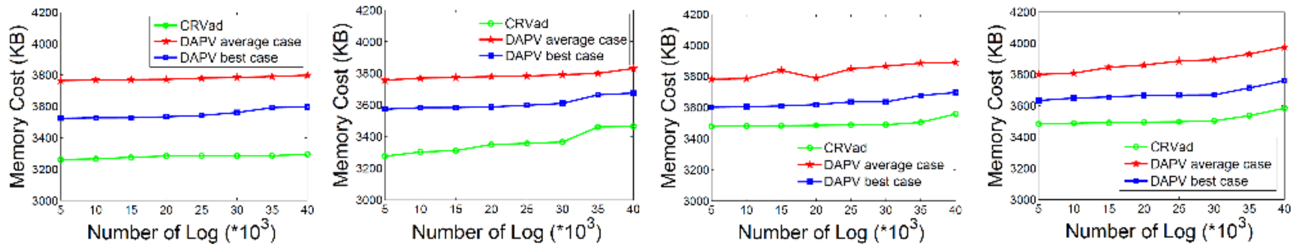
In the MANETs routing environment, the malicious nodes may exist in different order in the route path. They may be close to the destination or close to the source node. In the direct attack detection, the closer a malicious node is to the destination, the fewer logs will be verified. The verification phase is strongly related to the log entry number. So the verifications over a suspicious node may have different results due to different sizes of the log entries. As we have mentioned in Section III-B1, the depth of the Merkle Hash Tree is also relevant to the message coding length. The longer the message coding, the more time will be paid during the tree building and MHT verification. In the routers, there are many logs that we should deal with. Some logs are needed for our deducing and verification and others are irrelevant. We show the performance that DAPV and CRVad deals with different sizes of logs.

We use NS-3 to simulate the ad hoc network environment and let the nodes send and receive packets. At the same time, the log file for each node will be created as a .pcap file. Each node reads the file and uses the information to build up the Merkle Hash Tree and conducts the verification phase. The time cost and memory cost of the detection, including



(a) (b) (c) (d)

FIGURE 7. Time cost of the verification. (a) Msg Code Len = 4. (b) Msg Code Len = 6. (c) Msg Code Len = 8. (d) Msg Code Len = 10.



(a) (b) (c) (d)

FIGURE 8. Memory cost of the verification. (a) Msg Code Len = 4. (b) Msg Code Len = 6. (c) Msg Code Len = 8. (d) Msg Code Len = 10.

the cost for constructing the MHT, are respectively given in Figure 7 and Figure 8. In these cases, the green line is our prior work where the CRVad which concentrates on detecting attacks when the routing path has been built. The red line indicates the average results of our DAPV which can detect the attacks in both the routing discovery phase and data forwarding phase. The blue line is the best case of the DAPV's performance during the attack detection. First, we can see from Figure 7 and 8 that time cost or memory increases as the message code length and number of logs increase. Second, DAPV has higher results in the time cost as well as the memory cost. The DSA and provenance graph introduced to DAPV increase the cost in time and memory. DAPV has more rules arranging from route discovery phase to the data forwarding phase deducing process than the CRVad, which also costs more time during the MTH building, log deducing and verification. The overall cost is limited and the results show that our approach is practical for use on real network logs.

C. USABILITY: BLACK HOLE ATTACK DETECTION

1) PERFORMANCE OF SINGLE BLACK HOLE DETECTION

In the single black hole detection, we measure the detection accuracy rate of CRVad, DAPV without log pruning and DAPV with log pruning. We also set the routing length from 2 hops to 12 hops. Figure 9 shows our result. With CRVad, the detection accuracy rate varies from 79% to 84.7% averaging 82%. As for DAPV without the log pruning, the accuracy rate is from 88% to 92.5% averaging 90.3%. The pruning log generally improve the accuracy rate by 6%. With the log pruning, DAPV achieves the accuracy rate from 96% to 98.5% averaging 97%. According to the result, with the log pruning, we can achieve a higher detection accuracy rate.

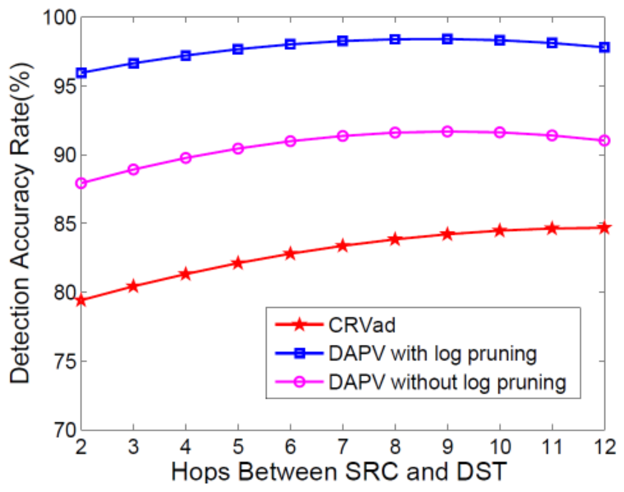


FIGURE 9. Detection accuracy rate on single black hole.

Figure 10 shows the time comparison among the original AODV protocol, DAPV without log pruning and DAPV with log pruning. Without the log pruning, DAPV needs to check more redundant logs and rules which will consume much more time than that with the pruning process. The result shows that the method without pruning can have the time cost from 148.2 ms to 223.8 ms which is relatively higher than the original AODV protocol. Without much abundant log comparison process, the method of DAPV with log pruning can bring the time cost down from 18.0 ms to 35.1 ms which should be acceptable in the MANETs.

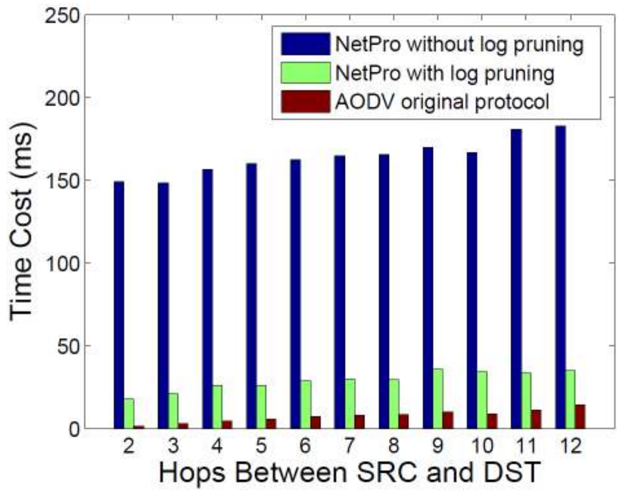
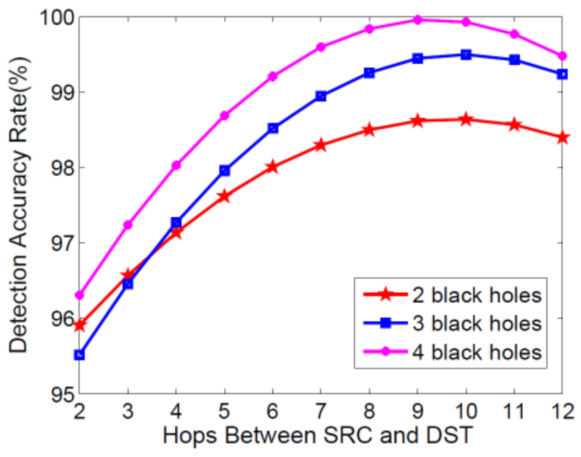
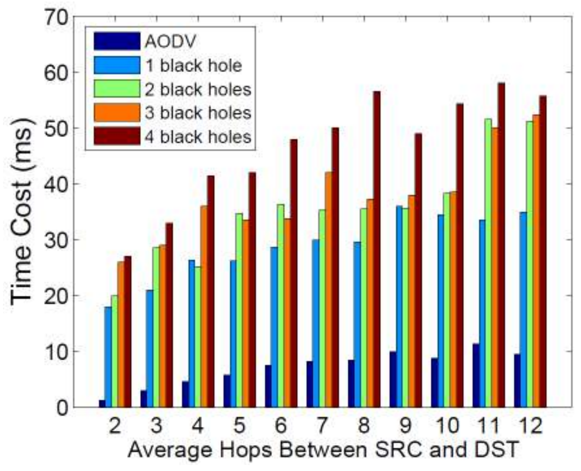


FIGURE 10. **Time cost of single black hole detection comparison.**



(a)



(b)

FIGURE 11. **DAPV performance on multi-black holes detection. (a) Detection accuracy rate on multi-black holes. (b) Time cost of multi-black holes detection comparison.**

2) PERFORMANCE OF MULTIPLE BLACK HOLES DETECTION

We apply multiple black holes into the MANETs and many of them cooperate with each other to launch the attack.

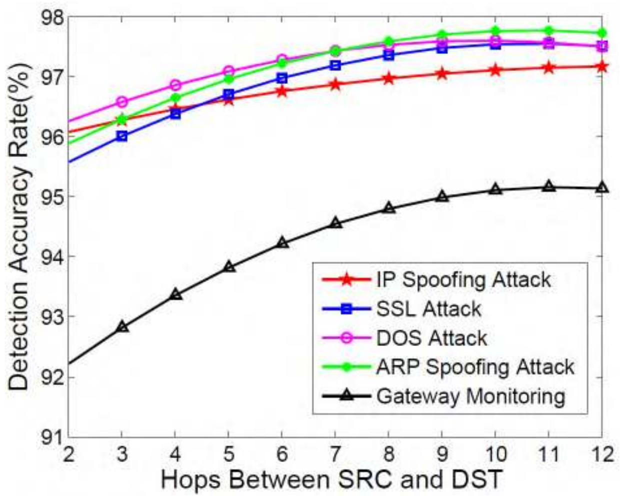


FIGURE 12. **DAPV detection accuracy rate on five attacks.**

We compare the results of DAPV from a single black hole environment to a 4 black holes environment and we are surprised to find out that our method had better results in handling the multiple black holes. When the coordinated malicious nodes show up, they cooperate with each other and help each other to deceive the source node. When the benign nodes find one of them, it is easy to find the nodes which help with the malicious nodes. The results in Figure 11(a) show that we gain a good result when the black holes arise. Figure 11(b) shows the time cost of multiple black holes detection and as the black hole increases the time cost also arises. But, we also notice that the detection accuracy rate drops from 10 hops to 12 hops in the multi-black holes detection. We still try to find out the reason and we assume it is relevant to the original AODV protocol's packet loss rate, because we find that the packet loss rate rises from 10 hops.

D. USABILITY: ROUTERS ANOMALIES DETECTION

We evaluate DAPV by using the real packets data from a university network from November 20th 2016 to April 20th 2017. We extracted the parameters from the packets and transformed them into the form of tuple mentioned in Table 1. The routers in the network include Cisco (CRV100, RV110W), Huawei (WS832, WS833) and TP-Link (TL-WR842N, TL-WR886N). We performed 5 different attacks towards routers with the tools and platforms shown in Table 3. Then, we evaluated the performance of DAPV on detecting these attacks.

We launched these five kinds of attacks on the path using the tools and platforms in the above table. We did each attack 20 times each month for every router in the path and calculated the average results according to Equation 17.

$$Average\ Accuracy\ Rate = \frac{\sum_{i=2}^{N} \frac{C_{n_i}}{Ts}}{N} \tag{17}$$

C_{n_i} represents the correct detection times of the total attack times (Ts is 100 in our experiment) and N means the number

TABLE 3. Attacks and the tools.

No.	Attack Name	Tool	Platform	Number of Attacks	Avg Attack Duration
1	IP Spoofing Attack	Nmap	Windows 7	125	6s
2	SSL Attack	THC-SSL	Windows 7	189	15s
3	DOS Attack	HULK	Ubuntu 12.07	167	2s
4	ARP Spoofing Attack	WinArpAttacker	Windows 7	134	9s
5	Gateway Monitoring Attack	WinArpAttacker	Windows 7	172	4s

of the routers in the path (the first router's i is 1 and the last is N). According to Figure 12, the detection accuracy rates of the first four attacks are approach each other and DAPV achieves a lower accuracy rate on the gateway monitoring attack. When we launched the first four attacks towards the routers, they could cause the routers to go down or not respond. Thus, these attacks will influence the forwarding function of the routers and DAPV can easily detect these anomalies during the reasoning process. As for the gateway monitoring attack, it will not influence the normal forwarding of the router and it is difficult to detect as the routers all behave normally. However, this kind of attack will influence the normal logs in the routers. Therefore, we used the logs in the routers to do the reasoning process and gained a good result in detecting this kind of indirect attack.

$$Precision = \frac{Correctly\ detected\ attacks}{Total\ inserted\ attacks} \tag{18}$$

$$Recall = \frac{Correctly\ detected\ attacks}{Total\ detected\ attacks} \tag{19}$$

We compare the performance of detecting these five attacks between DAPV and CRVad. We inserted these 5 kinds of attacks respectively in the routing paths which range from 2 hops to 12 hops. The precision and recall are calculated according to Equation 18 and 19. We collected the key-value pairs of recall and precision. Then, we select the points which range from 0.1 to 1 in the recall and draw the fitted curve in Figure 13. The error sum of the squares of DAPV and CRVad are 0.0151 and 0.0144 respectively, which are acceptable in our experiments. CRVad only detects the attackers in the routing path which ignores the routers that may be under attack by the malicious nodes. DAPV can detect the anomalies that contain the malicious nodes and the paralyzed routers that are attacked and can achieve better results than CRVad.

V. RELATED WORK

A. PROVENANCE

There is substantial literature on tracing provenance in databases [11] and in networks [12], [25], but only a few papers consider both positive and negative provenances in the routing mechanism. SNP [13] focuses on explaining to their operators why the network systems are in a certain state. ExSPAN [29] enables provenance support in the database system and SPIDeR can verify the route decision procedure. None of these papers consider the problem of negative provenance. Wu *et al.* [30] provided a method to answer why-not

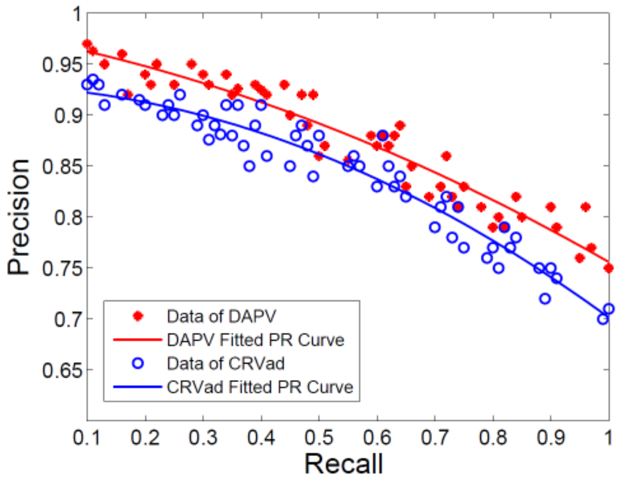


FIGURE 13. Comparison between DAPV and CRVad on Precision and Recall.

queries in SDN with negative provenance and the following work Y! [17] can also track the negative provenance in SDN and BGP. Both of these works should involve human operators and cannot start automatically and the distributed system scenario is the arbitrary control environment which can share the information among the nodes without considering privacy. None of these works considers the distributed environment and networks, as we do here. In the network literature, there is some prior work on tracking the faults in the routing system, including our own work CRVad [26], but it can only guarantee the security when the routing link path has already been built.

B. PRIVACY

This line of work, initiated by PeerReview [6] which detects faults by collecting all of the observations about each node violating our privacy policy. NetReview proposes to use the hash chain to verify the correctness of the BGP. Papadimitriou *et al.* [21] used this reasoning combined with the zero knowledge map to verify the node's correctness, but trusted the MOS(minimal observer set) as credible information which can be malicious at the first step. In recent work, such as Y! [17], SNP [13], MCI [31] and meta provenance [32], they can give back trace map towards the misbehavior but they seldom consider privacy preserved in the distributed system. None of these works can achieve both privacy preservation as well as credible verification as we do in our work.

C. ATTACK DETECTION

In the context of ad hoc networks, there are a lot of studies focusing on attack prevention, such as [33]–[35] which mainly focus on the security issues before the link was built, but most of them are secure network protocols and they cannot detect some malicious behavior in both the routing discovery phase and data forwarding phase. Several secure protocols [36]–[38] have been proposed to guarantee the security of the logs, but they require that nodes meet a certain criteria sharing a common secret key and only focus on one kind of attack which is either direct or indirect attack, and cannot tackle both of them. Our method is to detect the malicious nodes during the routing discovery phase and data forwarding phase as well as detect direct and indirect attacks. NetPro [39] and SRDPV [40] can mitigate the above problems but it ignores the paralyzed routers in the path attacked by remote adversaries. DAPV cannot only diagnose the malicious nodes who launched the attacks in the path but also spot the paralyzed routers which are attacked by the adversaries that may not be in the route path.

VI. CONCLUSION

In this paper, we proposed an approach, DAPV, to automatically detect the direct and indirect attacks by using provenance in MANETs. First, we used NDlog in reasoning the expected log and then checking whether the destination has been influenced. Next, we conducted the direct attack detection or the indirect attack detection with the assistance of MHT to preserve privacy. Finally, DAPV returned a provenance graph to show the tracking trace to the malicious nodes. DAPV could explain why an event was at fault and why an expected event did not occur with the concept of provenance. According to the algorithm of DAPV, we could detect which router was malicious and what kind of attack it was without revealing any privacy information. According to our experiment, our approach is scalable and practical for use on real MANETs routing security. Most of the debugging systems based on the log entries assume that the logs of each node truthfully reflect its behaviors. The logging system is independent from the communication mechanisms that malicious attackers may affect. Our system can contribute to solving the distributed verification and anomaly detection in Unmanned Aerial Vehicle networks, the 5th generation networks and the sensor networks. They all need to achieve the verification in a distributed way and without a third party. For the practical use of our approach in the future, rigorous specifications on the logging actions are needed, and code-level verification is indispensable to ensure the logging behavior under the constraint of these specifications.

REFERENCES

[1] L. Abusalah, A. Khokhar, and M. Guizani, "A survey of secure mobile Ad Hoc routing protocols," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 78–93, 4th Quart. 2008.

[2] A. Nadeem and M. P. Howarth, "A survey of MANET intrusion detection prevention approaches for network layer attacks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2027–2045, 4th Quart. 2013.

[3] H. Deng, W. Li, and D. P. Agrawal, "Routing security in wireless ad hoc networks," *IEEE Commun. Mag.*, vol. 40, no. 10, pp. 70–75, Oct. 2002.

[4] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet leashes: A defense against wormhole attacks in wireless networks," in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun.. IEEE Societies*, vol. 3, Apr. 2003, pp. 1976–1986.

[5] T. Li, J. Ma, and C. Sun, "Dlog: Diagnosing router events with syslogs for anomaly detection," *J. Supercomputing*, vol. 74, no. 2, pp. 845–867, Feb. 2018.

[6] A. Haeberlen, P. Kouznetsov, and P. Druschel, "PeerReview: Practical accountability for distributed systems," *ACM SIGOPS Symp. Oper. Syst. Princ.*, vol. 41, no. 6, pp. 175–188, 2007.

[7] A. Haeberlen, I. C. Avramopoulos, J. Rexford, and P. Druschel, "Netreview: Detecting when interdomain routing goes wrong," in *Proc. NSDI*, Apr. 2009, pp. 437–452.

[8] J. Jiang, W. Li, J. Luo, and J. Tan, "A network accountability based verification mechanism for detecting inter-domain routing path inconsistency," *J. Netw. Comput. Appl.*, vol. 36, no. 6, pp. 1671–1683, Nov. 2013.

[9] Y. Xie, V. Sekar, M. K. Reiter, and H. Zhang, "Forensic analysis for epidemic attacks in federated networks," in *Proc. IEEE Int. Conf. Netw. Protocols*, Nov. 2006, pp. 43–53.

[10] P. Laskowski and J. Chuang, "Network monitors and contracting systems: Competition and innovation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 183–194, Sep. 2006.

[11] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Database Theory—ICDT*. New York, NY, USA: Springer, 2001, pp. 316–330.

[12] Y. Liu *et al.*, "Towards a timely causality analysis for enterprise security," in *Proc. 25th Netw. Distrib. Syst. Secur. Symp. (NDSS)* San Diego, CA, USA: The Internet Society, 2018, pp. 1–15.

[13] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proc. 33rd ACM Symp. Operating Syst. Princ.*, Oct. 2011, pp. 295–310.

[14] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "I know what your packet did last hop: Using packet histories to troubleshoot networks," in *Proc. 11th USENIX Symp. Networked Syst. Design Implement.*, Sep. 2014, pp. 71–85.

[15] A. J. Gurney, A. Haeberlen, W. Zhou, M. Sherr, and B. T. Loo, "Having your cake and eating it too: Routing security with privacy protections," in *Proc. 10th ACM Workshop Hot Topics Netw.*, Nov. 2011, p. 15.

[16] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[17] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 383–394, Aug. 2015.

[18] W. El-Hajj, "The most recent SSL security attacks: Origins, implementation, evaluation, and suggested countermeasures," *Secur. Commun. Netw.*, vol. 5, no. 1, pp. 113–124, Jan. 2012.

[19] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A system for denial-of-service attack detection based on multivariate correlation analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 447–456, Feb. 2014.

[20] S. Y. Nam, S. Djuraev, and M. Park, "Collaborative approach to mitigating ARP poisoning-based Man-in-the-Middle attacks," *Comput. Netw.*, vol. 57, no. 18, pp. 3866–3884, Dec. 2013.

[21] A. Papadimitriou, M. Zhao, and A. Haeberlen, "Towards privacy-preserving fault detection," in *Proc. 9th Workshop Hot Topics Dependable Syst.*, Nov. 2013, p. 6.

[22] B. T. Loo *et al.*, "Declarative networking," *Commun. ACM*, vol. 52, no. 11, pp. 87–95, May 2009.

[23] D. Williams and E. G. Sirer, "Optimal parameter selection for efficient memory integrity verification using merkle hash trees," in *Proc. 3rd IEEE Int. Symp. Netw. Comput. Appl.*, Sep. 2004, pp. 383–388.

[24] A. R. Yumerefendi and J. S. Chase, "The role of accountability in dependable distributed systems," *HotDep*, vol. 5, p. 3, Jun. 2005.

[25] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao, "Efficient querying and maintenance of network provenance at Internet-scale," in *Proc. ACM SIGMOD Int. Conf. Manage. data*, Jun. 2010, pp. 615–626.

[26] T. Li, J. Ma, and C. Sun, "CRVad: Confidential reasoning and verification towards secure routing in ad hoc networks," in *Algorithms Architectures for Parallel Processing*. New York, NY, USA: Springer, 2015, pp. 449–462.

[27] NS-3. *Ns-3 Software*. Accessed: 2019. [Online]. Available: https://www.nsnam.org/

[28] IRIS. *Iris Reasoner*. Accessed: 2018. [Online]. Available: http://iris-reasoner.org/
[29] W. Zhou *et al.*, "Nettrails: A declarative platform for maintaining and querying provenance in distributed systems," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2011, pp. 1323–1326.
[30] Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "Answering why-not queries in software-defined networks with negative provenance," in *Proc. 10th ACM Workshop Hot Topics Netw.*, Nov. 2013, p. 3.
[31] Y. Kwon *et al.*, "MCI: Modeling-based causality inference in audit logging for attack investigation," in *Proc. 25th Netw. Distrib. Syst. Secur. Symp. (NDSS)*. San Diego, CA, USA: The Internet Society, 2018, pp. 25–39.
[32] Y. Wu, A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "Automated bug removal for software-defined networks," in *Proc. NSDI*, Sep. 2017, pp. 719–733.
[33] R. Hauser, T. Przygienda, and G. Tsudik, "Lowering security overhead in link state routing," *Comput. Netw.*, vol. 31, no. 8, pp. 885–894, Apr. 1999.
[34] D. B. Johnson, "Routing in ad hoc networks of mobile hosts," in *Proc. 1st Workshop Mobile Comput. Syst. Appl.*, Dec. 1994, pp. 158–163.
[35] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," Mobile Comput., Boston, MA, USA, Tech. Rep. RFC 3561, 2003.
[36] D. B. Johnson, "The dynamic source routing protocol for mobile ad hoc networks," Mobile Comput., Boston, MA, USA, Tech. Rep. RFC 4728, 2003.
[37] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. B. Royer, "A secure routing protocol for ad hoc networks," in *Proc. 10th IEEE Int. Conf. Netw. Protocols*, Nov. 2002, pp. 78–87.
[38] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *Proc. SCS Commun. Netw. Distrib. Syst. Model. Simul. Conf. (CNDS)*. San Antonio, TX, USA, Jan. 2002, pp. 193–204.
[39] T. Li, J. Ma, and C. Sun, "Netpro: Detecting attacks in manet routing with provenance and verification," *Sci. China Inf. Sci.*, vol. 60, no. 11, 2017, Art. no. 118101.
[40] T. Li, J. Ma, C. Sun, and N. Xi, "SRDPV: Secure route discovery and privacy-preserving verication in MANETs," in *Wireless Networks*. Nov. 2017, pp. 1–17.

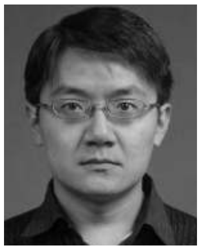
TENG LI received the B.S. and Ph.D. degrees from the School of Computer Science and Technology, Xidian University, China, in 2013 and 2018, respectively, where he is currently a Lecturer with the School of Cyber Engineering. His current research interests include wireless and mobile networks, distributed systems, and intelligent terminals, with the focus on security and privacy issues.

JIANFENG MA received the B.S. degree in computer science from Shaanxi Normal University, in 1982, the M.S. degree in computer science from Xidian University, in 1992, and the Ph.D. degree in computer science from Xidian University, in 1995, where he is currently a Professor with the School of Computer Science and Technology. He has published over 150 journal and conference papers. His research interests include information security, cryptography, and network security.

QINGQI PEI received the B.Sc., M.Sc., and Ph.D. degrees from Xidian University, Xi'an, China, in 1998, 2004, and 2008, respectively, where he is currently a Professor and a Ph.D. student supervisor. His research interests mainly focus on wireless communication networks and security, and information security. He has published over 90 papers in the significant international journals or conferences, and has granted 47 patents. He is a member of the ACM, a Senior Member of the Chinese Institute of Electronics, and a Senior Member of the China Computer Federation.

HOUBING SONG (M'12–SM'14) received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, USA, in 2012. In 2017, he joined the Department of Electrical, Computer, Software, and Systems Engineering, Embry-Riddle Aeronautical University, Daytona Beach, FL, USA, where he is currently an Assistant Professor and the Director of the Security and Optimization for Networked Globe Laboratory. His research interests include cyber-physical systems, cybersecurity and privacy, the Internet of Things, edge computing, big data analytics, unmanned aircraft systems, connected vehicle, smart and connected health, and wireless communications and networking.

YULONG SHEN received the B.S. and M.S. degrees in computer science and the Ph.D. degrees in cryptography from Xidian University, Xian, China, in 2002, 2005, and 2008, respectively, where he is currently a Professor with the School of Computer Science and Technology. He is also an Associate Director of the Shaanxi Key Laboratory of Network and System Security, and a member of the State Key Laboratory of Integrated Services Networks, Xidian University. His research interests include wireless network security and cloud computing security. He has also served on the Technical Program Committees for several international conferences, including the ICEBE, INCoS, CIS, and SOWN.



SUN CONG received the Ph.D. degree in computer science from Peking University, in 2011. He is currently an Associate Professor with the School of Cyber Engineering, Xidian University. His research interests include information flow security and program analysis.