JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 943-960 (2018)
DOI: 10.6688/JISE.201807_34(4).0009

PVad: Privacy-Preserving Verification for Secure Routing in Ad Hoc Networks

TENG LI, JIAN-FENG MA, CONG SUN AND NING XI
School of Cyber Engineering
Xidian University
Shaanxi, 710071 P.R. China
E-mail: litengxidian@gmail.com

Routing security has a great importance to the security of Mobile Ad Hoc Networks (MANETs). There are various kinds of attacks when establishing the routing path between the source and destination. The adversaries attempt to deceive the source node and get the privilege of data transmission. Then, they try to launch the malicious behaviors such as passive or active attacks. Due to the characteristics of the MANETs, *e.g.* dynamic topology, open medium, distributed cooperation, and constrained capability, it is difficult to verify the behaviors of nodes and detect malicious nodes without revealing any privacy. In this paper, we present PVad, an approach conducting privacy-preserving verification in the routing discovery phase of MANETs. PVad tries to find the existing communication rules via the association rules instead of making the rules. PVad consists of two phases, a reasoning phase deducing the expected log data of the peers, and a verification phase using a Merkle Hash Tree to verify the correctness of the derived information without revealing any privacy of nodes on the expected routing paths. Without deploying any special nodes to assist the verification, PVad can detect multiple malicious nodes by itself. To show that our approach can be used to guarantee the security of the MANETs, we conducted our experiments in NS3 as well as the real router environment, with the improvement of the detection accuracy by 4% on average compared to our former work.

Keywords: MANETs, detection, verification, privacy, diagnostics

1. INTRODUCTION

The Mobile Ad hoc Networks (MANETs) [1] are continuously self-configuring, infrastructure-less networks of mobile devices connected without wires. In such a network, all mobile nodes collaborate with each other and establish routing in a self-organized way. The primary goal of routing in MANETs is to establish a correct and efficient path through which data can be efficiently and securely transmitted. The adversaries may launch attacks in the routing discovery phase and perform malicious behaviors after they get involved in the routing path. If the routing is initialized incorrectly or the messages are forwarded through malicious adversaries, the entire network will be prone to be paralyzed.

To mitigate the security problems in MANETs, the prospective secure routing mechanism of MANETs should also consider the following features. *Self-organized.* Because there is no centralized node in MANETs, each node acts as both the host and router [2]. It is necessary to have a decentralized verification to check the security in MANETs without introducing a third party. *Dynamic topology.* Dynamic topology causes wireless links to be established and broken immediately. An adversary can send spu-

Received August 29, 2017; revised November 1, 2017; accepted December 12, 2017.
Communicated by Xiaohong Jiang.

rious messages to the source node to sneak into the routing path. To protect MANETs from the attacks, it is significant to impose a security mechanism on the routing discovery phase. *Coordinated attack*. The adversaries launch attacks in a coordinated way, on the same or different routing paths. Recent efforts that focus on detecting attackers on the same routing path [3] become insufficient. A new strategy to deal with multiple-path malicious behaviors is needed. *Privacy Preserving.* There is an inherent tension between verification and privacy-preservation because the attack detection usually requires revealing private log data of nodes [4, 5]. Recent work, such as PeerReview [6], detects faults by collecting all information from each node, which cannot satisfy the privacy-preserving requirement. Most importantly, we need a solution in MANETs to perform the verification without revealing important private data.

Considering the above features, we present PVad (Privacy-Preserving Verification for Secure Routing in Ad Hoc Networks), a mechanism which guarantees the security of Ad Hoc Networks during the routing discovery phase. First, PVad can supervise MANETs to choose a secure routing path from dynamically changing topologies in the routing discovery phase. Second, PVad conducts verification through the hosts in a self-organized manner without any help from a third party. The decentralized nodes cooperate with each other to derive evidence from important routing logs according to the predefined reasoning rules. After establishing a secure path and ensuring destination as a legal host, PVad can detect the existence of malicious nodes on the routing path as well as the exact position of them. Finally, PVad does not compromise our goal of protecting privacy for each node. PVad combines reasoning and verification to combat adversaries depending on the log evidence that we have already learned during the execution phase, and we needn't merge the whole log table from different routers.

The rest of the paper is organized as follows. We provide the overview and roadmap of our approach in Section 2. Then, we elaborate the reasoning and verification of PVad in Section 3. In section 4, we present our implementation and evaluation that consists of both NS3 simulations and real case studies. We depict related work in Section 5. Finally, we conclude the paper in Section 6.

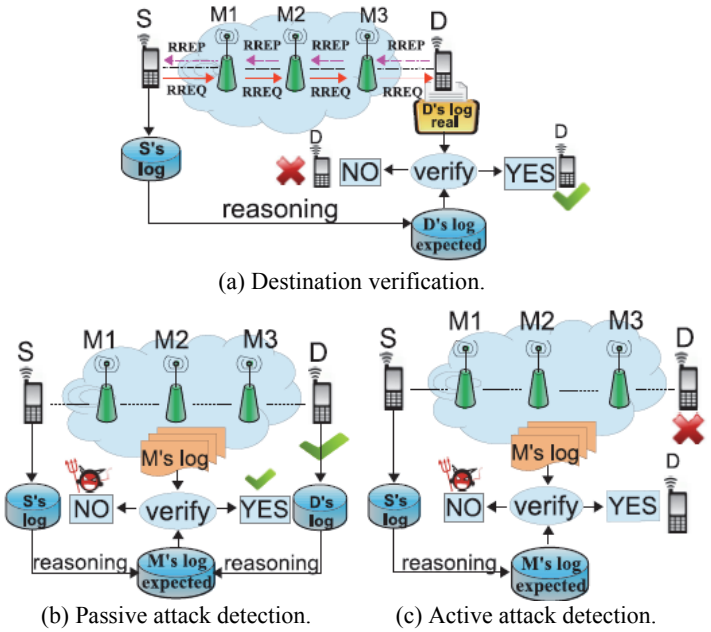
2. OVERVIEW AND ROADMAP

We consider an ad hoc network scenario in which the nodes connect with each other randomly. Only the message source is initially trusted and there might be more than one malicious node during the routing path discovery phase.

In the routing discovery phase, the source node sends a routing request RREQ to its neighbors [7]. The IP address of destination is mounted in the RREQ packet. Whenever an intermediated node has the correct routing or it is the exact destination, it will reply with a routing response RREP to the source node. Then, a routing path is established and the sender will start data transmission through the established path. In the attacker scenario, malicious nodes can send fake packets back to the source node in order to defraud the right of routing data during the routing discovery [8]. For instance, a malicious node can declare that it has the shortest and latest path to the destination by forging the RREP packet, which is known as the first step of the black hole attack.

Therefore, the source node should verify the authenticity of the destination. Alt-

hough the routing path to the real destination has been established, we cannot ensure whether the routing is expedited or if the destination received the messages the source sent in case of intermediated malicious nodes. PVad can verify that each node participating in data transmission along the routing path is benign using the reasoning rules for transmission and the privacy-preserving evidence from each node. Otherwise, a potential attack may occur at some malicious nodes. The overview architecture of PVad is given in Fig. 1.



(a) Destination verification.

(b) Passive attack detection. (c) Active attack detection.

Fig. 1. The procedure of PVad.

Unlike ProTracer [9] that logged the information by its own debugging system, we make use of logs that exist in the routers. First, we use the logs on the trusted source node *S* to deduce the information that should be held by destination node *D*. We are not sure whether *D* is benign or it received the messages we sent. PVad verifies *D's* real log to check whether it can provide evidence to prove its correctness by using the Merkle Hash Tree. Fig. 1 (a) shows the destination verification. If the result is true, we use the information from the source and destination to verify whether the intermediated nodes are well-behaved (Fig. 1 (b)). If the result is false, we just use the log of the destination to deduce the logs of the intermediated routers (Fig. 1 (c)). In this way, we can finally guarantee the security of the intermediated routers along the routing path and make sure the whole path is healthy.

With PVad, we can first establish the correct routing path and find out the real destination. Then, we can ensure the established routing path is secure and the malicious intermediated nodes are detected. Unlike the previous study on minimum observer sets [10] which trusts both the source and destination to deduce the information of the intermediated routers, we choose to trust only the source node because we cannot get the in-

formation from the real destination from the beginning. The attacker model is more general to allow the destination node to be malicious.

Except for guaranteeing the security of the routing process of MANETs, this paper makes the following contributions:

1. Accuracy. Our approach should not regard a normal node as a malicious one, and we can detect the exact malicious nodes as well as the general features of the attacks.
2. Early-protection. The mechanism should not realize that there are malicious nodes in the routing path when some problems emerge during the data transmission or the network is paralyzed. The detection is performed at the earliest routing discovery phase.
3. Non-third-party. The whole confidential reasoning and hash verification should be accomplished without introducing any extra nodes or third party.
4. Privacy-preserving. The system should not reveal the confidential logs directly. Any attempt to leak the confidentiality of the nodes is forbidden.

3. PVAD REASONING AND VERIFICATION

In this section we elaborate PVad in detail. We explain the confidential reasoning as well as the privacy-preserving verification of PVad.

3.1 Finding the Rules in MANETs

In MANETs, the equipments link and communicate with each other in certain ways which have been set before they send the messages. We need to find these communication rules instead of making them. Many approaches [11, 12] relied on rule-based processing which improved the routers' debugging ability, but it requires the operator to have domain knowledge and involves a human operator to make the rules for the detection. Making the rules for the working routers is impossible in MANETs, and PVad tries to find the existing rules for them. We use the Apriori algorithm to do the rule-mining among the routers and we study the basic communication rules in the following section.

Association rules describe items that occur together frequently in a dataset and are widely-used for market basket analysis. As the original definition [13], the problem of association rule-mining is defined as: Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of attributes called *items*. For example, *sendRequest*(@*C*, *S*, *D*, *SEQ*) can be one item. Let $D = \{t_1, t_2, ..., t_m\}$ be a set of transactions called the *database*. Each transaction in D contains a subset of items in I. A *rule* is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \in I$ and $X \cap Y = \emptyset$. Support $supp(X)$ of an itemset X is defined as the proportion of transactions in the data set which contains the itemset. The confidence of a rule is defined as

$$conf(X \Rightarrow Y) = \frac{supp(X \cap Y)}{supp(X)}. \quad (1)$$

In our problem setting, each log message is one item. We get the logs from the .pcap file of NS-3 [14] and the syslogs from the routers. In order to construct the transactions, we use a sliding window W and the size of the window will influence the performance of

finding rules. We discuss the window size in our evaluation part. The logs in the same time window are regarded as a transaction. In one such transaction, the log messages in window W are considered as the items showing up. We find the rules of the routing discovery phase in MANETs through the association rules and we set the threshold of the confidence as 100%. $conf(X \Rightarrow Y)$=100% means that if X emerges, Y will also exist.

3.2 The Deduction Rules for PVad

We use NDlog to conduct the reasoning procession. NDlog is based on the Datalog [15]. A Datalog program includes a set of declarative rules. Every rule has the form p :- q_1, q_2, q_3, ..., q_n ..., which means as "q_1, q_2, q_3, and q_n implies 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. Datalog rules can refer to one another in a recursive fashion. NDlog supports the location specifier as a store of information in each rule. The specifier is represented as an @ symbol followed by an attribute.

We conduct the subsequent confidential deduction by using the rules in Table 1. In these rules, C and R represent the sender's and receiver's storage place respectively. S represents the sender, D means the receiver and SEQ represents the destination sequence number which is used to determine the freshness of the routing information. R is the storage place of the receiver and the intermediated router stores its messages in the place of M. MSG is the message that the source node intends to send to the destination host. In the phase of the routing discovery, MSG can be the HELLO message or other test message to check whether the routing path is unblocked.

When a sender is to find a route, it first broadcasts a routing request, *sendRequest* (@C, S, D, SEQ), to its neighbors. The request means S wants to find a routing path to D whose sequence number is in SEQ and this request is logged at C. When the neighbors get the request, *getRequest*(@M, S, D, SEQ), they compare their own IP with the received one; if two values are different, the neighbors continue to broadcast the routing request *reqForward*(@M, S, D, SEQ, $STAUS$). But if the two values are the same, this means the destination is found, and the destination sends the reply to the source node *sendReply*(@R, S, D, SEQ) (Rules 1-3). When the source node gets the reply and the destination gets the request, it means that we found the destination (Rule 4). The predicate *findDest* and *findDest'* have identical semantics, but the difference is only for the storage place (Rule 5). Once the sender finds the destination, the data link is established (Rule 6). After that, when the sender proposes the message sending request *msgRequest*(@C, S, D, SEQ, MSG), the destination should authorize the request (Rule 7). After authorization, the source node begins to send the messages and the intermediated routers should forward the messages (Rule 8).

Except for the primary rules presented in Table 1, there are still auxiliary NDlog rules to support the correct running of our whole reasoning process. For instance, due to Rule 4 in Table 1, we know the following auxiliary rule holds: *getRequest*(@R, S, D, SEQ):- *findDest*(@C, S, D, SEQ), *getReply*(@C, S, D, SEQ). We know Rule 1, and we also imply that the following rule holds: *sendRequest*(@C, S, D, SEQ):- *getRequest*(@M, S, D, SEQ).

Table 1. The deduction rules for PVad.

1. getRequest(*@M*, *S*, *D*, *SEQ)*	:- sendRequest(*@C*, *S*, *D*, *SEQ*)
2. reqForward(*@M*, *S*, *D*, *SEQ*, *STAUS*)	:- getRequest(*@M*, *S*, *D*, *SEQ*) *STAUS*='*YES*/*NO*'
3. getReply(*@C*, *S*, *D*, *SEQ*)	:- reqForward(*@M*, *S*, *D*, *SEQ*, *STAUS*) sendReply(*@R*, *S*, *D*, *SEQ*) *STAUS*='*NO*'
4. findDest(*@C*, *S*, *D*, *SEQ*)	:- getReply(*@C*, *S*, *D*, *SEQ*) getRequest(*@R*, *S*, *D*, *SEQ*)
5. findDest'(*@M*, *S*, *D*, *SEQ*)	:- findDest(*@C*, *S*, *D*, *SEQ*)
6. dataLink(*@C*, *S*, *D*, *SEQ*, *STAUS*)	:- findDest(*@C*, *S*, *D*, *SEQ*) sendRequest(*@C*, *S*, *D*, *SEQ*) *STAUS*='*YES*/*NO*'
7. authSend(*@R*, *S*, *D*, *SEQ*, *MSG*)	:- dataLink(*@C*, *S*, *D*, *SEQ*, *STAUS*) msgRequest(*@C*, *S*, *D*, *SEQ*, *MSG*) *STAUS*='*YES*'
8. msgForward(*@M*, *S*, *D*, *SEQ*, *MSG*)	:- authSend (*@R*, *S*, *D*, *SEQ*, *MSG*) sendMsg(*@C*, *S*, *D*, *SEQ*, *MSG*)

3.3 Reasoning

With the above deduction rules, we can initiate the whole reasoning process. First, we use the sender's information to infer messages from the receiver to identify the destination node. For convenience, this phase is denoted by $S \Rightarrow D$ (source node deduces the destination node), which consists of the reasoning of the following propositions:

sendReply(*@C*, *S*, *D*, *SEQ*) $\wedge$ sendRequest(*@C*, *S*, *D*, *SEQ*) $\rightarrow$sendReply(*@R*, *S*, *D*, *SEQ*)	(r1)
getReply(*@C*, *S*, *D*, *SEQ*) $\wedge$ msgRequest(*@C*, *S*, *D*, *SEQ*) $\rightarrow$sendReply(*@R*, *S*, *D*, *SEQ*)	(r2)
dataLink(*@C*, *S*, *D*, *SEQ*, '*YES*') $\wedge$ msgRequest(*@C*, *S*, *D*, *SEQ*, *MSG*) $\rightarrow$authSend(*@R*, *S*, *D*, *SEQ*, *MSG*)	(r3)

We use the messages of *S* stored at the place of *C* to infer information from the destination stored at *R*. Combined with the basic rules and NDlog language, the expected information of the receiver can be obtained. These messages should be on the destination node according to the transmission mechanism. Then, a major problem is how we know whether the receiver truly has such information. One simple solution is that we can go through the log information of *D* and then we will know the result. However, this approach can violate the principle of privacy and we cannot achieve verifiability. Therefore, how can we conduct a privacy-preserving verification which protects the privacy of an individual log at the same time? Next, we will introduce the Merkle Hash Tree into our method to realize the goal of privacy protection.

3.4 Privacy-Preserving Verification

In this section, we introduce the mechanism of privacy-preserving verification. We use the Merkle Hash Tree [16] (MHT) to preserve the privacy during verification. MHT is a tree in which every non-leaf node is labeled with the hash value of the labels of its

children nodes, and every leaf node is labeled with the hash value of real data. As a kind of binary tree, the edges of MHT from every parent's node to its two children are tagged with 0 and 1 respectively; see Fig. 2. The purpose of verification is to ensure that the expected actions or messages derived in the reasoning phase exist in the log of related nodes. The process consists of four steps: encoding messages, tree building, hash calculating, and final verification.

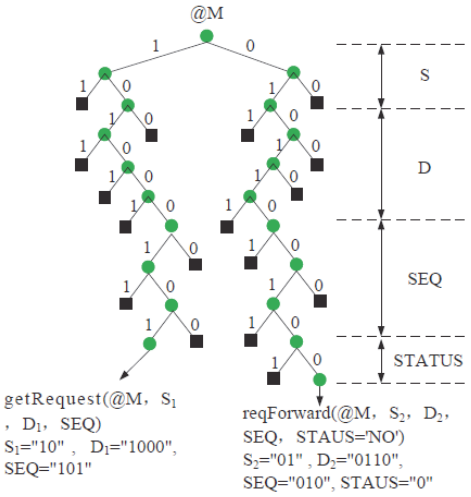


Fig. 2. Merkle hash tree.

Before building the tree, we should first encode the information. Take the following message encoding as an example. *getRequest*(*@M*, S_1, D_1, *SEQ*) can be encoded by specifying S_1="10", D_1="1000", *SEQ*="101". The message *reqForward*(*@M*, S_2, D_2, *SEQ*, *STAUS*='*NO*')can be encoded by specifying S_2="01", D_2="0110", SEQ="010", STAUS (NO)="0". How many bits the variables, such as S_1 and S_2, cost for the message encoding will depend on the number of nodes and number of variables. Our goal is to distinguish each message for different nodes by encoding the messages.

Next, it is the tree building process. After encoding the text message *reqForward* (*@M*, S_2, D_2, *SEQ*, *STAUS*='*NO*') as a string "0101100100", we use it to build the tree. Each router will build a Merkle Hash Tree using its log information. We label the node's left child as 1 and right child as 0. With the string, we can build the tree, which is stored as an array.

Then, we can calculate the hash value of each node *i*: $H_i = H(\text{node\_num} \,||\, \text{bit\_data} \,||\, \text{parent\_num} \,||\, \text{string} \,||\, \text{parent\_bit\_data} \,||\, H_{\text{left\_child}} \,||\, H_{\text{right\_child}})$. As an index of node array, *node_num* specifies the array element w.r.t the current node. Similarly, *parent_num* specifies the array element w.r.t the parent of the current node. *bit_data* and *parent_it_data* are the value of the current node and parent node respectively, which can be either 0 or 1. When a node is the root of MHT, we define *parent_bit_data* of this node as 'X'. $H_{\text{left\_child}}$ and $H_{\text{right\_child}}$ are the hash value of the left child and right child respectively. They will be empty if the current node is a 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_2 infers the information *reqForward* (@*M*, S_2, D_2, *SEQ*, *STAUS*='*NO*') of D_2, then S_2 will ask D_2 whether you are the real destination with such information. To answer the question, first, D_2 uses the string "0101100100" to find the leaf node in its own hash tree and then provides the leaf's brother hash value and the *node_num*, *parent_num*, *parent_bit_data* and *string* of the leaf for S_2 during every level of the tree. Then S_2 can use these values to calculate the root hash of the tree from the leaf to the root. If the latter equals the published value, the verification result is true.

After we have ensured that the destination node is the legal host, we move to the intermediated router verification phase. To make sure the whole routing path is healthy without malicious nodes, we need to verify the forwarding nodes, ensure that they transmit the data according to the rules and eliminate the possibility of launching attacks such as passive or active attacks. We have proved that the information of the destination node is correct, and we can use the information on both *S* and *D* to conduct the following deduction. We call this process as $S+D \Rightarrow M$ (source node and destination node deduce intermediated routers). In this process we should reason the following propositions:

sendRequest(@*C*, *S*, *D*, *SEQ*)→getRequest(@*M*, *S*, *D*, *SEQ*)	(r4)
sendReply(@*C*, *S*, *D*, *SEQ*) ∧ getRequest(@*R*, *S*, *D*, *SEQ*) →reqForward(@*M*, *S*, *D*, *SEQ*, '*NO*')	(r5)
getReply(@*C*, *S*, *D*, *SEQ*) ∧ getRequest(@*R*, *S*, *D*, *SEQ*) →findDest'(@*M*, *S*, *D*, *SEQ*)	(r6)
sendMsg(@*C*, *S*, *D*, *SEQ*, *MSG*) ∧ authSend(@*R*, *S*, *D*, *SEQ*, *MSG*) →msgForward(@*M*, *S*, *D*, *SEQ*, *MSG*)	(r7)

Then, we use the Mekle Hash Tree to verify that the real logs of the intermediated routers comply with the expected log messages, *e.g. reqForward* (@*M*, *S*, *D*, *SEQ*, *STAUS*).

If the logs of the destination are false, we should find out which intermediated routers influenced them and find out the passive attacker. We call this process as $S \Rightarrow M$ (source node deduces the intermediated routers). In this process, we should reason the following propositions:

sendRequest(@*C*, *S*, *D*, *SEQ*)→reqForward(@*M*, *S*, *D*, *SEQ*)	(r8)
sendRequest(@*C*, *S*, *D*, *SEQ*)→getRequest(@*M*, *S*, *D*, *SEQ*, *STAUS*)	(r9)
findDest(@*C*, *S*, *D*, *SEQ*)→findDest'(@*M*, *S*, *D*, *SEQ*)	(r10)
dataLink(@*C*, *S*, *D*, *SEQ*, '*YES*')∧sendMsg(@*C*, *S*, *D*, *SEQ*, *MSG*) ∧msgRequest(@*C*, *S*, *D*, *SEQ*, *MSG*) →authSend(@*R*, *S*, *D*, *SEQ*, *MSG*)	(r11)

3.5 PVad Algorithm

We define *S* as the sender and *D* as the receiver, and m_i represents the medial routers (m_1, m_2, ..., m_n). The functions *Reason*() and *MHT*() in Algorithm 1 are used to reason the expected logs and verify the logs.

Step 1: The sender uses its own logs to reason the expected logs (τ'_D) of the destination with the rules. Then, the sender verifies the expected logs and the real logs of the destination with MHT. If the verification result is false, PVad jumps to step 3 for active attack detection. Otherwise, PVad jumps to step 2 for passive attack detection.

Step 2: We use the log of *S* and *D* to reason the prospective results about the log of the intermediated routers m_i, and use the results to verify the real log of m_i with the Merkle Hash Tree. If a violation is found, there exist some faults and the router m_i has the malicious behavior. If all routers R_i are well-behaved, then the routing path was chosen.

Step 3: We use the log of *S* to reason and verify the logs of the intermediated routers to find out the passive attackers.

Algorithm1: PVad Algorithm

Input: *intermediated_route_list*(m_1, m_2, m_3, ..., m_n) contains intermediated routers; τ_C, source's logs; τ_D, destination's logs. *Reason*(t_C, *D*) is the function that using the logs(τ_C) on the sender to deduce the logs on the destination (*D*). *MHT*(t_D, t'_D) is the function that using the MHT to verify the real logs and the expected logs.

Output: Return the detected malicious node.

```
1.  τ'_D=Reason(τ_C, D)
2.  result = MHT(τ_D, τ'_D);
3.  if result then
4.     "D is in the good condition"
5.     for i = n to 1 do
6.        τ'_i = Reason(τ_C + τ_D, m_i) ;
7.        result=MHT(τ_i, τ'_i);
8.        if !result then
9.           mi changed its own logs;
10.       else
11.          CONTINUE;
12.       end if
13.    end for
14. else
15.    for i = n to 1 do
16.       τ'_i = Reason(τ_C, m_i) ;
17.       result=MHT(τ_i, τ'_i);
18.       if result then
19.          BREAK;
20.       else
21.          CONTINUE;
22.       end if
23.    end for
24.    if i == 0 then
25.       return "D is the malicious node"
26.    else
27.       return "m_i is the malicious node"
28.    end if
29. end if
```

4. EVALUATION

In this section, we evaluate PVad through our experimental results. Specifically, our goal is to answer the following research questions.

4.1 Experiment Setup

We did our experiments on NS-3 and a real router environment respectively. In

NS-3, we configured the nodes that ran the AODV protocols and we injected few malicious nodes that dropped or tampered with the logs of the nodes. As for the real routers, we launched attacks towards the routers in the link. To evaluate PVad's performance, we set our former work CRVad [17] and SyslogDigest [18] as the benchmarks. CRVad does not use the algorithm to find the rules and is handled by operators, while SyslogDigest focuses on the syslog analyses to do the anomaly detection. We show the performances of our approach and the results of the comparison with these two works in this section.

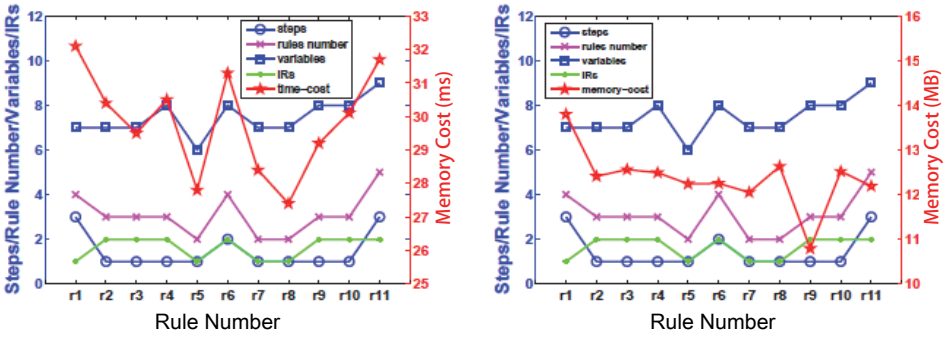
4.2 Description of System Models

PVad uses an open source tool IRIS [19] to conduct the reasoning process, and proves that the reasoning rules we inferred are correct. All of the reasoning results we used have been written in NDlog and verified by IRIS. We build up multi-hop topology among different nodes. In the simulation environment, we use the Logging Module of NS3 to log sent and received messages. We use the tracking system in NS3 to get the content of the log. In a real case environment, we get the syslogs from the real routers and store them in the SQL server. Then, each node can use these kinds of log entries to build up its own Merkle Hash Tree.

PVad can detect the misbehavior and identify the malicious nodes automatically. First, PVad extracts the logs from .pcap files or SQL server. Second, Pvad uses the Apriori algorithm to do the rule-mining among the logs. Third, for the source node, PVad uses its logs with the corresponding rules to deduce the expected logs of the target nodes (with the open source tool IRIS). Next, combining this with MHT, the source node conducts provenance verification to confirm whether the real log entries match the expected ones. Finally, according to our algorithm, PVad returns the exact malicious nodes.

4.3 Factors Influencing the Reasoning Efficiency

We use IRIS to conduct our reasoning process. We first write the reasoning rules in NDlog programs, and then extract the sending/receiving evidence from the logs. According to this evidence, the prospective truths which should be provided by the suspicious nodes are reasoned out by using IRIS. The time and spatial cost of the reasoning of each rule are given in Fig. 3.



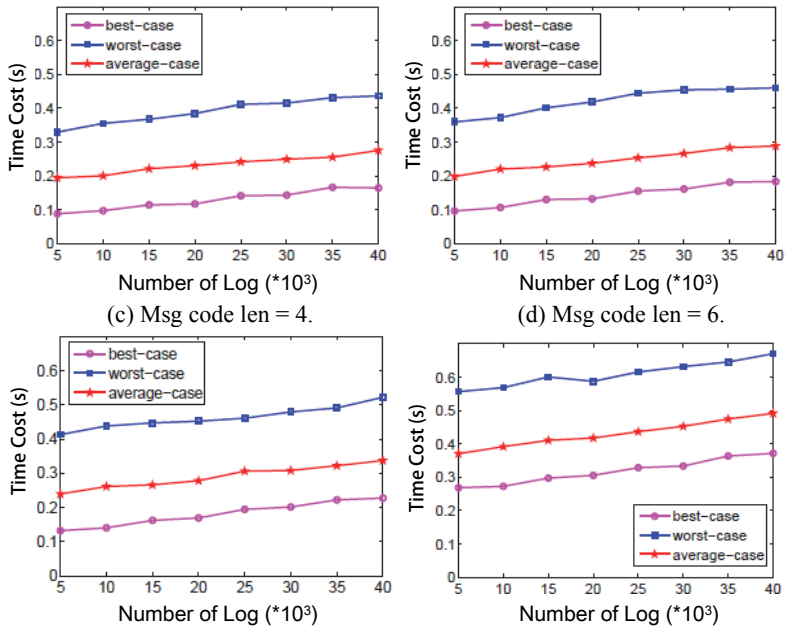
(a) Time cost of the reasoning. (b) Memory cost of the reasoning.

Fig. 3. The performance of reasoning.

The evidence which should exist in the nodes is reasoned before verification and we just confirm this evidence to judge which node committed the attacks. We have observed three main factors, which are reasoning steps, variables of the tuples and initial rules, affecting the time and memory cost of PVad. From Fig. 3, we can see that the results of the time and memory cost vary with the changing of these factors especially with the variables of the tuples. As for the time cost, we can see strong relevance of the time cost and variables from r1-r8. The reasoning steps and IRs (Initial Rules) have little influence on the time costs. In memory cost, we can also see that the fewer the steps, the higher the memory costs from r4-r5. According to the analysis, the variable number is the most significant factor among these three parameters. The whole time and spatial cost is acceptable according to our experimental results.

4.4 Performance on Different Orders of Verification

In the wireless environment, the malicious nodes can behave differently for different routing paths. They may act as a normal node on one path, but pretend to be a wanted destination on another path after intercepting the reply message. Thus, the verifications over a suspicious node may have different results. Only when we confirm that the node is malicious for the first time, we can decide that the routing path with the malicious node is unavailable. Therefore, the verification results vary randomly when we perform the verification on different routing paths in different orders. If we can detect the malicious node at the beginning, only a single path is verified. In the worst case, we should check all of the potential routing paths to detect the maliciousness of the node. We also evaluate the average cost when the verification order is randomly selected.

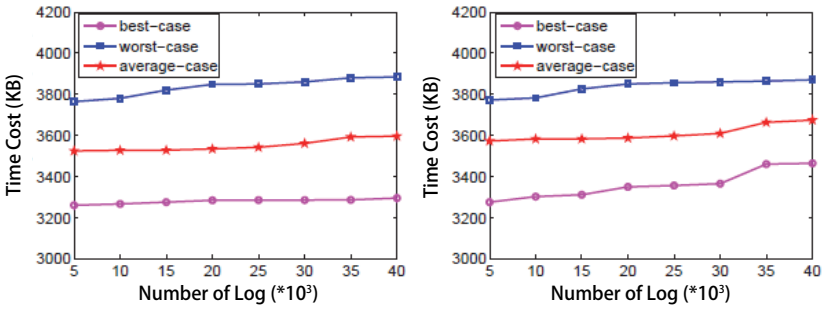


(c) Msg code len = 4. (d) Msg code len = 6.

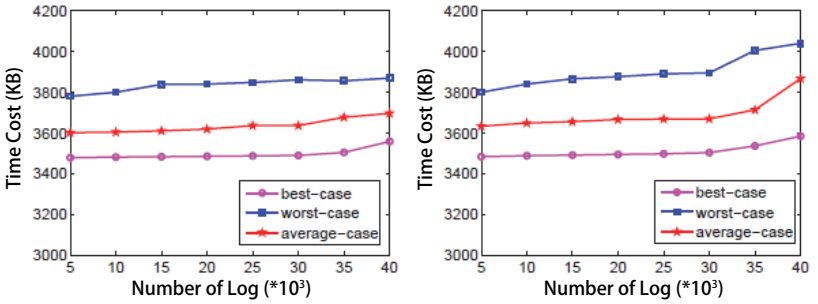
(c) Msg code len = 8. (d) Msg code len = 10.

Fig. 4. Time cost of the verification.

We use NS3 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 verification, including the cost for constructing the MHT, are given in Figs. 4 and 5 respectively. The best-case scenario is when we only verify the destination once and then find the malicious node immediately. The worst-case scenario means we verify the destination according to different routing paths and find the malicious node last. The average-case scenario means that we get the results of time and memory costs from general verification. First, we can see from the results that time cost or memory increases as the message code length and number of logs increase. Second, single verification can have a significant reduction in the time cost as well as the memory cost. The overall cost is limited and the results show that our approach is practical for use on real network logs.



(c) Msg code len = 4. (d) Msg code len = 6.



(c) Msg code len = 8. (d) Msg code len = 10.

Fig. 5. Memory cost of the verification.

4.5 Time Window Size Influencing the Accuracy of the Apriori Algorithm

During the rule learning phase, we need to collect the logs of different routers in MANETs. Because the log entries have inherent time sequences, we should set the time window size when we use the Apriori algorithm. In our experiment, we set the window size from 0s to 120s and we compare the results we obtained with the former setting rules with the following equation:

$$Accuracy = \frac{Correctly\ extracted\ rules}{Total\ extracted\ rules}. \quad (1)$$

When PVad learns the association rules, we need to set the time window size to separate the log chunks. PVad runs the Apriori algorithm on the logs within the same window size. Due to the results in Fig. 6, time window size influences the accuracy rule learning. From the results, we achieved best accuracy rate among 50s to 80s and the log entries increase dramatically after the time window of 80 seconds.

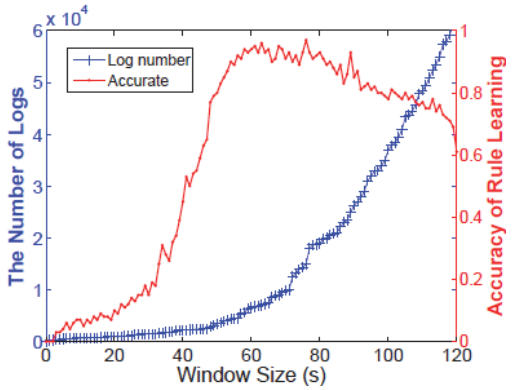


Fig. 6. Time window influences on Apriori accuracy.

4.6 Performance of Detecting Attacks on Real Routers

We also launched our experiment on the real routers that consist of Huawei, Cisco, and Dlink. We launched an IP Spoofing Attack, SSL Attack, DOS Attack, ARP Spoofing Attack, and Gateway Monitoring Attack towards the routers and these attacks can influence the logs on the routers. We compared the results of PVad with our former work CRVad [17] and SyslogDigest [18], which can also verify the correctness of the nodes in MANETs. The precision and recall are calculated according to Eqs. (2) and (3). 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 Fig. 7.

$$Precision = \frac{Correctly\ detected\ attacks}{Total\ inserted\ attacks} \tag{2}$$

$$Recall = \frac{Correctly\ detected\ attacks}{Total\ detected\ attacks} \tag{3}$$

As the Fig. 7 shows, PVad can achieve a better result than CRVad. The error sum of the squares of PVad, CRVad and SyslogDigest are 0.0068, 0.0144 and 0.0257 respectively, which are acceptable in our experiments. CRVad used the rules that were set by the operators and it required the help of human experts to deal with these rules. Therefore, in the real router environment, it can reduce the performance of the detection accuracy. SyslogDigest used the subtype tree to extract the templates from the raw logs and learnt the events by using the association rule. We apply SyslogDigest to extract the event and use the PCA algorithm [20] to detect the anomalies. During the construction of the subtype tree, it is difficult for SyslogDigest to choose a proper parameter in pruning

the tree branches [21]. PVad mitigated this problem by using the machine learning algorithm and it can learn the rules that existed in the Network. Besides, we construct the log entries into the Merkle Hash Tree for verification and it can preserve the privacy of the nodes, which do not have the problem of choosing the parameter in pruning the tree branches as SyslogDigest. PVad can be more accurate than the two methods as shown in our results.

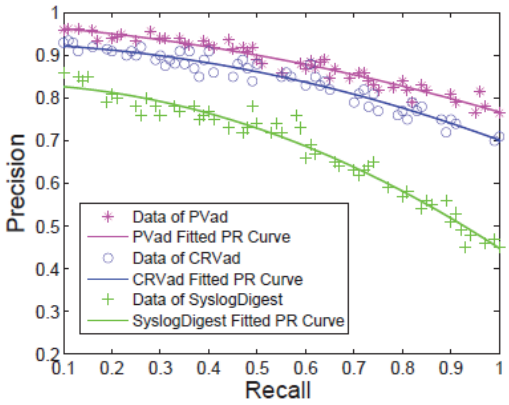


Fig. 7. Comparison between PVad, CRVad and SyslogDigest on PR.

5. RELATED WORK

Reasoning: There is substantial work on reasoning attacks or bugs in the database [22] and networks [23], but only a few papers consider the reasoning in a distributed and automatic way for the routing field. SNP [12] explains its operators as to why the network systems are in a certain state. ExSPAN [24] provides reasoning in the database system. None of these papers consider the problem why such state or log entries do not exist or are missed. Wu *et al.* [25] provide a method which answers why-not queries in SDN with a negative reasoning link. The following work Y! [26] can also track the negative reasoning link in SDN and BGP. Both of the works should involve human operators but do not start automatically. SDN can get the information by the controller which can collect the information among 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, such as CRVad [17]. However, it cannot help the source avoid the malicious node in the routing discovery phase.

Privacy: This line of work, started by PeerReview [6] detects faults by collecting all observations about each node. There is an inherent tension between verification and privacy-preservation because the verification usually requires revealing private log entries or packets of nodes [4]. To solve the privacy issue, NetReview [27] proposed to use the hash chain to verify the correctness of the BGP and Hayajneh *et al.* [28] proposed a lightweight public key authentication scheme for MSN systems. Papadimitriou *et al.* [10] used reasoning combined with the zero knowledge proof to verify the correctness of the nodes instead of trusting the MOS (minimal observer set) as credible information. How-

ever, the approach can be influenced by the malicious destination at the first step. In recent work, such as Y! [26] and SNP [12], they can provide the tracing link to the misbehavior but they seldom consider the privacy preservation in the distributed system. None of these works can achieve both privacy preservation as well as credible verification like what we do in our work.

Network diagnostics: In the context of MANETs, there are a lot of papers focusing on attack detection, such as [29, 30] which mainly solve the security issues after the link is built, but most of them cannot prevent some malicious behavior in the routing discovery phase. Several secure protocols [31, 32] have been proposed to guarantee the security of the log, but they require that all nodes meet a certain criteria sharing a common secret key and focus on only one kind of attack which is either an active or passive attack, but cannot tackle both of them. Nakayama *et al.* [33] proposed an approach to detect the malicious node based on dynamic learning, but their method should gather all of the nodes' information and finish the task in a central way. A few existing systems, such as PeerPressure [34], EnCore [35], ClearView [36] and Shen *et al.* [37], used statistical analysis or data mining to learn correct configuration values, performance models, or system invariants. But none of them accurately capture the causality of the states of the nodes or leverage causality to detect attacks. Our method is used to detect the malicious nodes in a distributed way during the routing discovery phase as well as detect active and passive attacks.

5. CONCLUSION

In this paper, we proposed an approach, PVad, to verify the nodes automatically without introducing a third party. We emphasized privacy preservation during the routing establishing phase. PVad uses the Apriori algorithm to find the communication rules in MANETs instead of making them. We used NDlog in reasoning expected information and a Merkle Hash Tree to preserve the confidentialities of the destination and the intermediated routers. According to PVad, we can eliminate both the fake destination and malicious intermediated routers. PVad achieves the goal of verifying the routing path in a self-organized manner without revealing any private data from the log of the nodes. Our approach is scalable and practical for use in real MANETs.

REFERENCES

1. L. Abusalah, A. Khokhar, and M. Guizani, "A survey of secure mobile ad hoc routing protocols," *Communications Surveys & Tutorials*, Vol. 10, 2008, pp. 78-93.
2. A. Nadeem and M. P. Howarth, "A survey of manet intrusion detection & prevention approaches for network layer attacks," *Communications Surveys & Tutorials*, *IEEE*, Vol. 15, 2013, pp. 2027-2045.
3. H. Deng, W. Li, and D. P. Agrawal, "Routing security in wireless ad hoc networks," *Communications Magazine*, Vol. 40, 2002, pp. 70-75.
4. 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 *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011, p. 15.

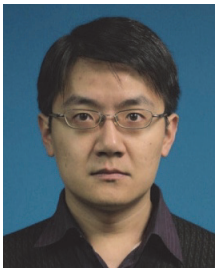
5. A. Chen, A. Haeberlen, W. Zhou, and B. T. Loo, "One primitive to diagnose them all: Architectural support for internet diagnostics." in *EuroSys*, 2017, pp. 374-388.
6. A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," *ACM SIGOPS Operating Systems Review*, Vol. 41, 2007, pp. 175-188.
7. S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *Proceedings of IEEE International Conference on Communications*, Vol. 10, 2001, pp. 3201-3205.
8. C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Technical Report, No. RFC 3561. 2003.
9. S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting," in *Proceedings of Network and Distributed System Security Symposium*, 2016, p. 15.
10. A. Papadimitriou, M. Zhao, and A. Haeberlen, "Towards privacy-preserving fault detection," in *Proceedings of the 9th ACM Workshop on Hot Topics in Dependable Systems*, 2013, p. 6.
11. Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," in *ACM SIGCOMM Computer Communication Review*, Vol. 44, 2014, pp. 383-394.
12. W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, 2011, pp. 295-310.
13. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM Sigmod Record*, Vol. 22, 1993, pp. 207-216.
14. NS-3, "ns-3 software," https://www.nsnam.org.
15. B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative networking," *Communications of the ACM*, Vol. 52, 2009, pp. 87-95.
16. D. Williams and E. G. Sirer, "Optimal parameter selection for efficient memory integrity verification using merkle hash trees," in *Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications*, 2004, pp. 383-388.
17. T. Li, J. Ma, and C. Sun, "Crvad: Confidential reasoning and verification towards secure routing in ad hoc networks," in *Proceedings of International Conference on Algorithms and Architectures for Parallel Processing*, 2015, pp. 449-462.
18. T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network: mining network events from router syslogs," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 472-484.
19. IRIS, "iris software," https://sourceforge.net/projects/iris-reasoner/.
20. H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, Vol. 2, 2010, pp. 433-459.
21. T. Li, J. Ma, and C. Sun, "Dlog: diagnosing router events with syslogs for anomaly detection," *Journal of Supercomputing*, Vol. 74, 2018, pp. 845-867.
22. P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Proceedings of International Conference on Database Theory*, 2001, pp. 316-330.

23. A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "The good, the bad, and the differences: Better network diagnostics with differential provenance," in *Proceedings of ACM Special Interest Group on Data Communication Conference*, 2016, pp. 115-128.
24. W. Zhou, Q. Fei, S. Sun, T. Tao, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr, "Net-trails: a declarative platform for maintaining and querying provenance in distributed systems," in *Proceedings of ACM International Conference on Management of Data*, 2011, pp. 1323-1326.
25. Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "Answering why-not queries in software-defined networks with negative provenance," in *Proceedings of the 12th ACM Workshop on Hot Topics in Networks*, 2013, p. 3.
26. Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," *ACM SIGCOMM Computer Communication Review*, Vol. 44, 2015, pp. 383-394.
27. A. Haeberlen, I. C. Avramopoulos, J. Rexford, and P. Druschel, "Netreview: Detecting when interdomain routing goes wrong," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009, pp. 437-452.
28. T. Hayajneh, B. J. Mohd, M. Imran, G. Almashaqbeh, and A. V. Vasilakos, "Secure authentication for remote patient monitoring with wireless medical sensor networks," *Sensors*, Vol. 16, 2016, p. 424.
29. M. Mohanapriya and I. Krishnamurthi, "Modified dsr protocol for detection and removal of selective black hole attack in manet," *Computers and Electrical Engineering*, Vol. 40, 2014, pp. 530-538.
30. W. Wang, G. Zeng, J. Yao, H. Wang, and D. Tang, "Towards reliable self-clustering mobile ad hoc networks," *Computers and Electrical Engineering*, Vol. 38, 2012, pp. 551-562.
31. W. Liu and M. Yu, "Aasr: authenticated anonymous secure routing for manets in adversarial environments," *IEEE Transactions on Vehicular Technology*, Vol. 63, 2014, pp. 4585-4593.
32. P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *Proceedings of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2002, pp. 193-204.
33. H. Nakayama, S. Kurosawa, A. Jamalipour, Y. Nemoto, and N. Kato, "A dynamic anomaly detection scheme for aodv-based mobile ad hoc networks," *IEEE Transactions on Vehicular Technology*, Vol. 58, 2009, pp. 2471-2481.
34. H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, Vol. 4, 2004, pp. 245-257.
35. J. Zhang, L. Renganarayana, X. Zhang, N. Ge, V. Bala, T. Xu, and Y. Zhou, "Encore: Exploiting system environment and correlation information for misconfiguration detection," *ACM SIGPLAN Notices*, Vol. 49, 2014, pp. 687-700.
36. J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan *et al.*, "Automatically patching errors in deployed software," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, 2009, pp. 87-102.
37. K. Shen, C. Stewart, C. Li, and X. Li, "Reference-driven performance anomaly iden-

tification," in *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37, 2009, pp. 85-96.

Teng Li (李腾) received the B.S. degree in Computer Science and Technology from Xidian University, China in 2013. He is currently working toward the Ph.D. degree at the school of computer science and technology, Xidian University, China. His current research interests include wireless and mobile networks, distributed systems and intelligent terminals with focus on security and privacy issues.

Jian-Feng Ma (马建峰) received the ME and Ph.D. degree in Computer Software and Communications Engineering from Xidian University, in 1988 and 1995, respectively. He is now a Full Professor and Ph.D. supervisor in Xidian University, member of China Computer Federation. His main research interests include information security, coding theory, and cryptography. He is a member of the IEEE.



Cong Sun (孙聪) received the Ph.D. degree in Computer Science from Peking University, in 2011. He is currently an Associate Professor in the School of Cyber Engineering at Xidian University. His research interests include information flow security and program analysis.

Ning Xi (习宁) received the B.S and M.S. degree in Computer Science and Technology from Xidian University, China in 2008 and 2011. And now he is a Ph.D. student in Computer Science and Technology, Xidian University. His major research is in heterogeneous network convergence, service computing and network security.