Detection of Attacks in Wireless Mesh Networks

Conference Paper · April 2011

DOI: 10.1109/LADC.2011.13 · Source: DBLP

CITATIONS
5

READS
340

2 authors:

Anderson Morais

21 PUBLICATIONS **162** CITATIONS

SEE PROFILE

Ana R. Cavalli

Institut Mines-Télécom

265 PUBLICATIONS **2,393** CITATIONS

SEE PROFILE

Detection of Attacks in Wireless Mesh Networks

Anderson Morais, Ana Cavalli
Software-Networks Department, Télécom SudParis
{anderson.morais, ana.cavalli}@it-sudparis.eu

Abstract

Wireless Mesh Network (WMN) is a new technology that is gaining importance among traditional wireless communication systems. Wireless Mesh Networks are becoming a reasonable choice to offer Internet access in an inexpensive, convenient, and quick way. However, WMNs are vulnerable to several kinds of attacks because of their inherent attributes such as the open communication medium. Malicious mesh devices can launch attacks to disrupt the network routing operations, then putting the entire mesh network at risk. This paper analyzes routing manipulation attacks against the proactive routing protocol Better Approach To Mobile Ad hoc Network (BATMAN). We demonstrate the feasibility of the attack in an emulated mesh network that consists of virtual nodes executing BATMAN protocol. Then, we show how to detect the attack by making use of traces produced by the mesh nodes.

1. Introduction

Wireless Mesh Network (WMN) has appeared in the wireless communication scenario as an emerging technology to fulfill the requirements of Next Generation Wireless Networks (NGWN), for offering adaptive, flexible, and reconfigurable network architecture while providing affordable solutions to wireless Internet Service Providers (ISPs) [1]. ISPs are using Wireless Mesh Network (WMN) to provide Internet connectivity since it allows an effortless, fast, and cost-effective network deployment.

WMNs are characterized by dynamic self-organization, self-configuration, and self-healing whereas the mesh nodes sets up an ad hoc network in an automatic way to maintain the connectivity between the nodes. The mesh nodes transmit traffic from others nodes to reach a destination that they could not reach by themselves. When a mesh node can no longer operate, its neighbors merely find another route through one or more intermediate nodes. Rather than traditional wireless networks, WMNs do not count on dedicated wireless infrastructure, but the nodes count on each other to maintain the network entirely connected. Consequently, the network is very reliable since there is frequently more than one route between a source node and a destination node.

WMNs can be considered as a class of ad hoc networks. Therefore, Mobile Ad hoc Networks (MANETs) and WMNs are equivalent, but MANETs also need to handle the issue of high mobility of nodes and frequent modifications of the network topology. There are basically two types of mesh nodes: mesh routers and mesh clients [2]. The mesh routers constitute a wireless mesh backbone among them and offer multi-hop wireless connectivity to the mesh clients. Mesh routers can be assigned as gateway routers, which are then connected to the Internet. The mesh routers have minimum mobility and perform the role of routing traffic transmitted from mesh clients, who normally do not have routing functions, to the gateway router, which is connected to the Internet. Wireless community networks and municipal wireless networks are good examples of real life WMNs, which offer low-cost Internet access via Wi-Fi to large areas by using inexpensive IEEE 802.11 wireless mesh routers.

WMNs are highly vulnerable to several types of attacks since they rely on shared wireless medium access and limited resources. Attacks against the network routing protocol, such as routing disruption and resource depletion, are particularly a serious issue in a mesh network because they can compromise the integrity of the routes and the availability of the entire network. Malicious nodes can generate malicious traffic, intercept and modify packets, or refuse to forward packets by dropping the packets. Even with end-to-end cryptographic protection, a malicious node can drop packets going through it, or manipulate the network routes by providing fake routing information.

In this paper, we introduce a routing manipulation attack targeting the Better Approach To Mobile Ad hoc Networking (BATMAN) routing protocol [3]. We demonstrate how a malicious node can manipulate the

routes of the mesh nodes and divert the nodes traffic to itself. To validate our analysis, we implement the attack in a virtualized mesh network that consists of virtual mesh nodes executing the BATMAN protocol. The malicious node behavior is emulated by a network tool. To detect the routing manipulation attack, we introduce an efficient approach that is based on the analysis of BATMAN traces.

The contributions of this paper are the analysis of the routing manipulation attack in a controllable virtualized network environment as the majority of ad-hoc network security approaches have been validated with simulators, and the proposed detection scheme to identify this kind of attack.

The remainder of the paper proceeds as follows. Section 2 describes the related work. Section 3 introduces BATMAN routing protocol. Section 4 provides a description of the routing manipulation attack. Section 5 presents the experiments with the attack. Section 6 describes the method proposed to detect the attack. Finally, section 7 concludes the paper.

2. Related work

A large number of studies have been conducted to conceive security methods for the existing routing protocols in MANETs. Various secure routing protocols have been designed such as ARAN [4], Ariadne [5], and SAODV [6]. The majority of approaches are based on key management or encryption technologies to authenticate the routing messages and assure that unauthorized nodes will not join the network. However, these schemes cannot avoid a compromised node, which has a legitimate key, or a non-authenticated node to launch attacks against the network nodes.

Although many security schemes have been proposed for WMNs in recent years, they are still not fully applicable for WMNs. In [7] and [8], the authors address the problem of privacy in WMNs. These approaches focus on traffic privacy by introducing anonymous routing algorithms to keep the routing information secret. Nevertheless, they do not tell how the mesh nodes implement authentication, key distribution, and key agreement. In [9], the authors introduce an active cache based defense mechanism in the mesh routers, which identifies flooding style DoS attacks in the network. They apply the most frequently used cache mechanism to detect such flooding of messages. In [10] a Secure Layer-2 Path Selection (SLPS) scheme is designed. It utilizes cryptographic extensions to offer authenticity and integrity of the routing messages in order to avoid attackers from manipulating the message fields. Nonetheless, SLPS protocol is vulnerable to message fabrication attacks.

The routing manipulation attack (also referred as link spoofing attack) has not been fully evaluated in WMNs. In [11], Kannhavong et al. describe a colluding link spoofing attack against Optimized Link State Routing (OLSR) [13] on a MANET. The attacker announces fake links to two-hop neighbors of the target node to divert the target node's traffic to itself. They implemented the attack in a simulator, and introduced a method to detect the link spoofing attack by using information of two hops neighbors. One downside to this scheme is that it could not be possible to detect link spoofing with nodes further away than three hops.

A location based detection technique, which detects link-spoofing attacks against OLSR, is presented in [12]. The method utilizes signatures along with an embedding GPS and a timestamp. The technique adds the geographical position and the timestamp of a node in control messages so that each node can know the current position of any other node in the network. The method detects the link spoofing by estimating the distance between two nodes that claim to be neighbors. The main limitation of this method is that it cannot be carried out in a scenario where a GPS device is installed on each node of the MANET.

3. BATMAN protocol overview

The majority of routing protocols employed in WMNs were designed based on routing protocols from ad hoc networks, which were conceived taking into consideration the nodes mobility. Common routing protocols such as OLSR [13], Dynamic Source Routing (DSR) [14], and Ad-hoc On-demand Distance Vector Routing (AODV) [15] are good examples. These ad hoc network protocols were developed based on the assumption that the network is regularly modifying its topology because of nodes mobility and signal losses over the wireless communication medium. Mesh networks can be seen as a special type of ad hoc network where little or no mobility is expected and only occasional route changes happen. BATMAN protocol attempts to maximize traffic throughput and minimize packet delay in the network instead of just keeping the connectivity among the nodes.

The BATMAN algorithm aims to share information of the best routes between nodes to all nodes in the mesh network. Each node only keeps the information of the best next-hop for every destination in the WMN. BATMAN algorithm is only concerned with learning about the best next-hop through which it can find the

best route since every node on the route knows the best next-hop that it should use to send messages. Therefore, it is not necessary to find or calculate the full route to all destination nodes in the mesh network.

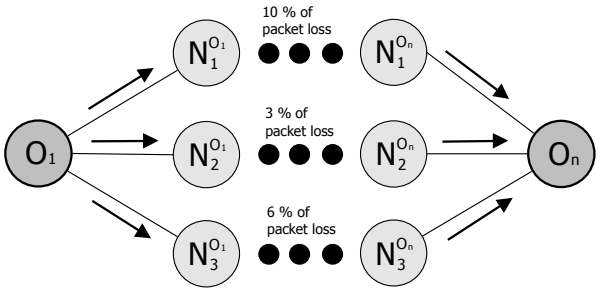
BATMAN implements a flooding mechanism based on *sequence numbers* that avoids propagating inconsistent topology information and avoids broadcasting an excessive number of topology messages through network. BATMAN algorithm was designed to deal with unreliable links in WMNs by performing statistical analysis of packet loss and propagation speed to select the most reliable route to the next-hop.

3.1. BATMAN flooding mechanism

The BATMAN algorithm is detailed as follows [16]. Each node (also known as Originator) periodically broadcasts hello messages, referred as Originator Messages (OGMs), to tell its neighbors about its existence. An OGM owns at least an *originator address*, a *sending node address*, a Time To Live (TTL), and a unique *sequence number* value. Following a neighboring node receives an OGM; it modifies the *sending node address* to its own address and rebroadcast this OGM in accordance to BATMAN forwarding rules to tell its neighboring nodes about the existence of the node that originated the OGM, and so on and so forth. Hence, the mesh network is flooded with OGMs until every node has received it at least once, or until they got lost because of packet loss of the wireless communication links, or until their TTL value has expired. The *sequence number* value of the OGM is utilized to verify how fresh the message is, i.e., to discern between new OGMs and duplicated OGMs to guarantee that each OGM is only counted once.

OGMs that travel on a bad route where the wireless link quality is poor will experience packet loss or delay on their path through the mesh network. On the contrary, OGMs that follow good routes will be transmitted faster and more reliable. The amount of OGMs, i.e., the total number of *sequence numbers*, from an Originator received via each neighboring node is utilized to calculate the route quality. To find out which is the best route to a destination node, the BATMAN node counts the number of OGMs (*sequence numbers*) originated by that node and received from its different neighboring nodes. Thus, BATMAN chooses as next-hop the neighboring node from which it has received the highest amount of OGMs within a *sliding window* (a packet count metric), i.e., the route owning the best quality. BATMAN makes use of this information to maintain a table containing the best next-hop towards every Originator, i.e., destination node, in the mesh network.

The BATMAN flooding mechanism is shown in Figure 1. In Figure 1 (a), the node O_n will know about the existence of the node O_1 in the distance by receiving node O_1's OGMs that are rebroadcasted by its neighboring nodes $N^{O_1}{}_1$, $N^{O_1}{}_2$, and $N^{O_1}{}_3$. Since node O_n has more than one neighboring node, it has to decide which node it will select as best next-hop to send data to O_1. This decision is made based on the number of OGMs it has received faster and more reliable via one of its neighboring nodes $N^{On}{}_1$, $N^{On}{}_2$, or $N^{On}{}_3$, as illustrated in Figure 1 (b). Then, node O_n choose node $N^{On}{}_2$ as its current best next-hop, i.e., best ranking neighbor, towards node O_1, and it updates its routing table respectively, as seen in Figure 1 (c).



(a) Mesh network.

Node	O_1	$N^{O1}{}_1$	$N^{O1}{}_2$	$N^{O1}{}_3$	$N^{On}{}_1$	$N^{On}{}_2$	$N^{On}{}_3$
Broadcasted OGMs	100	-	-	-	-	-	-
Rebroadcasted OGMs	-	100	100	100	-	-	-
Received OGMs	-	-	-	-	90	***97***	94

(b) OGMs broadcasted by node O_1.

Originator	Next-hop	Potential next-hop
O_1	$N^{On}{}_2$	$N^{On}{}_3$

(c) Routing table of node O_n.

Figure 1. BATMAN flooding scheme.

3.2. BATMAN advanced

BATMAN advanced (also known as *batman-adv*) is the BATMAN routing protocol implemented as a Linux kernel module executing on MAC Layer [16]. The majority of ad hoc network routing protocols implementations execute on Layer 3, which means they use UDP packets to send routing information, and manipulate the kernel routing table to update the routing information.

In *batman-adv*, the routing information and the data traffic are transmitted using raw Ethernet frames. *Batman-adv* encapsulates and forwards all data traffic

until it reaches the destination, thus emulating a virtual network switch of all mesh nodes. Hence, all nodes seem to be link local and not aware of the network's topology. The advantages of this design are the user can use other protocols such as IPv4 and IPv6 on top of *batman-adv* protocol, a node can participate in the mesh network without having an IP address, and mobile nodes can conveniently be attached to the mesh network.

4. Routing manipulation attack

This section describes the routing manipulation attack against BATMAN, where a malicious node disrupts the network routes by violating the integrity of the nodes' routing tables. We use state transition diagram to describe the attack.

4.1. Description of routing manipulation attack

A malicious node can generate and broadcast OGMs for a mesh node with continuous valid *sequence numbers*, which it actually did not receive, so it can control the routes of the target nodes, and divert the route to the destination to pass through it [16]. As routing decisions are based on statistical analysis of the amount of OGMs received instead of information contained in the packets, the attacker needs to generate a number of OGMs that are numerous enough to redirect the routes of target nodes to the malicious node. In other words, the attacker has to continuously win the neighbor ranking of the target nodes towards the destination, so prevailing over the legitimate OGMs. The attacker can apply two methods to execute the attack:

- ***Attacking method 1*: update the neighbor ranking range of the target nodes**. In this method, the attacker sends phony OGMs for an Originator containing *sequence number* values out of the *sliding window* size, i.e., the value range of the *sequence numbers* received so far. Thus, the target nodes will presume that the Originator has reinitiated and will update their neighbor ranking range. The attacker must constantly generate OGMs with valid *sequence numbers* to persuade all target nodes to update their neighbor ranking range each time they receive a phony OGM. If the *sliding window* of the link via which the attacker's OGMs have been received has the most *sequence numbers*, then this link is considered to be the best link to the Originator, i.e., to the compromised node.

- ***Attacking method 2*: not update the neighbor ranking range of the target nodes**. In this method, the attacker generates fake OGMs for an Originator with *sequence number* values a few counts ahead of the real *sequence number* value. These OGMs will be preferred in the ranking range, and they will not persuade the target nodes to update their neighbor ranking range. Nonetheless, the target nodes must broadcast a sufficient amount of fake OGMs to win the ranking neighbor.

The route manipulation attack is exemplified in Figure 2. It is assumed that node A is the attacker, and nodes T_1, T_2, T_3, and T_4 are the target nodes. Nodes T_1 and A are the neighboring nodes of the gateway node O_1. As the link between nodes O_1 and T_1 has a packet loss rate lower than the one between nodes O_1 and A, the target nodes T_2, T_3, and T_4 will choose a route via node T_1 toward node O_1.

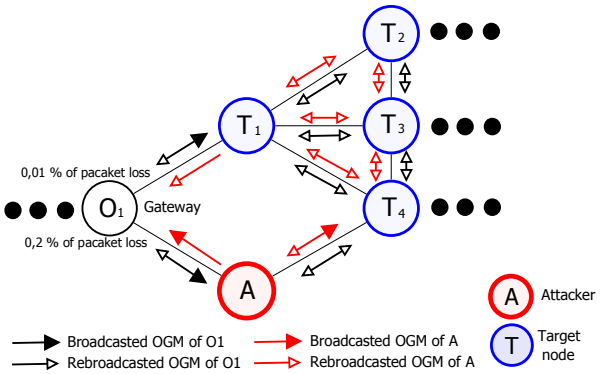


Figure 2. Route manipulation attack.

Then, the compromised node A generates and broadcasts OGMs for Originator O_1 with continuous valid *sequence numbers*, therefore applying the attack method 2, in order to redirect the route of the target nodes toward node O_1 to node A. The broadcasting rate of the fake OGMs is higher than BATMAN default rate defined for the nodes, thus the attacker always wins the ranking towards the destination.

In accordance with BATMAN algorithm, the target nodes will choose the compromised node A as the best next-hop to node O_1 because node A has rebroadcasted the most number of OGMs from node O_1. Since node A is the only best next-hop to the gateway node O_1, it can execute the following misuses:

- To discard data packets or OGMs that go through it. For example, node A can discard all OGMs from target node T_4. If node T_1 crashes, the node T_4's route entry will be deleted from node O_1's routing table, then node O_1 will not find a route to node T_4.

- To modify data packets or OGMs that pass through it. For example, node A can modify the destination

address of data packets from node T_4 to misguide the other mesh nodes to forward the node T_4's packets to unknown destinations.

4.2. Model of route manipulation attack

Figure 3 illustrates the extended state machine diagram of the route manipulation attack.

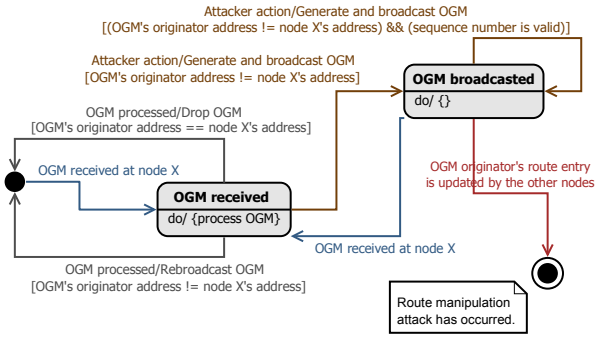


Figure 3. State machine for the attack.

The state machine is triggered when an OGM is received, and it moves to the initial state *OGM received*. Thus, *process OGM* action is executed, and then three transitions can be activated by *OGM processed* trigger, but two of them return to the initial state. The first transition, which returns to the initial state, has its guard condition evaluated as true if the OGM's originator address is the same as the node's address that received the OGM. That means this OGM was originated by the own node, then *Drop OGM* action is executed. In the second transition, the guard condition evaluates to true if the OGM's originator address is not the same as to the node's address. That means the OGM was received from a neighbor, then the transition executes *Rebroadcast OGM* action. The third transition of *OGM received* state moves to *OGM broadcasted* state if *attacker action* trigger is fired. In this transition, if the OGM's originator address is not the same as the node's address, the *Generate and broadcast OGM* action is executed.

When *OGM broadcasted* state is reached, if the attacker keeps generating and broadcasting OGMs with the originator address not same as the node's address, the state transition is constantly fired going back to *OGM broadcasted* state. At last, the second transition of *OGM broadcasted* state is fired if the nodes of the mesh network have updated their best next-hop towards the Originator that is being broadcasted by the attacker. Then, the state machine goes to the final state, which means that the route manipulation attack has been succeeded.

5. Experiment and results

The routing manipulation attack was implemented in a virtualized network environment, which emulates the mesh network. The mesh network is emulated by virtual machines executing the BATMAN protocol, which define the mesh nodes. The virtualized architecture setup was used:

- An "Intel(R) Xeon(R) CPU W3550 @ 3.06GHz" machine with 12,0GB of RAM executing a 64-bit Ubuntu Linux OS with Kernel 2.6.35 is used to run the instances of QEMU emulator [18], which represent the virtual nodes.
- Each QEMU instance executing a 32-bit Ubuntu Linux OS with kernel 2.6.35 is attached to a virtual switch (*vde_switch* tool [19] with some modifications [17]), and connected to each other by the *wirefilter* tool [19].
- The network configuration uses a fixed topology, where each virtual node is connected to a virtual switch, which transmits packets if they came from the node's virtual neighbors. In that way, the links between the nodes could be emulated. The packet loss is emulated by the *wirefilter* tool.

Batman-adv v.2010.1.0 [17] was compiled from the source code and loaded into each virtual node's Linux kernel. BATMAN executes on only one network interface per virtualized node. The default *originator interval* value of 1000 ms is used, which is value in milliseconds that defines how frequently BATMAN broadcast OGMs.

We used *batman-adv* event logging and debugging output mechanisms to obtain all BATMAN debugging messages. This debugging information is saved to log files that are synchronized by a global clock, i.e., the time of the main machine when it launches the virtual machines. During the debugging data collecting at each virtual node, the main machine average CPU usage was 40% and the main memory usage was 60%, thus there was no performance degradation and resource contention to interfere in the collected results.

We also changed packETH tool [20] (an Ethernet packet generator) in order to add a new functionality which generate and sends OGMs with continuous valid *sequence numbers* values on the virtual node's network interface. Therefore, we can emulate the attacks by executing the modified tool on the compromised node.

We simulated the attack in two different network scenarios. In the first scenario, we added one attacker in a smaller virtual network, and in the second scenario, we added two attackers in a more complex network topology.

5.1. Execution of the attack at scenario 1

The emulated mesh network of this scenario has seven mesh nodes, as illustrated in Figure 4, which comprises one attacker A_2, one gateway node O_1, and five target nodes T_3, T_4, T_5, and T_6. The attacker A_2 generates and broadcasts phony OGMs for node O_1 to divert the routes of the target nodes toward node O_1 to node A_2.

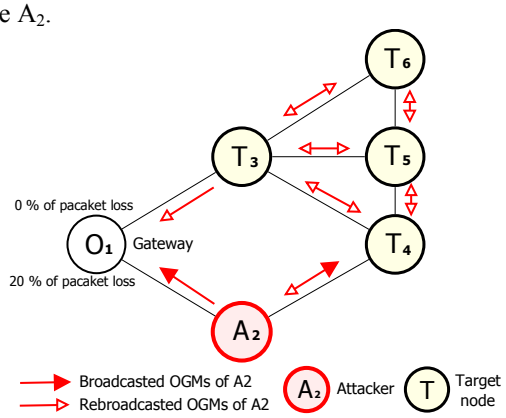


Figure 4. Mesh network topology for scenario 1.

The duration of each experiment is 20 minutes. Each virtual node takes about 4 minutes for starting up, configuring its virtual connections, and establishing the routes with the other nodes. After that, the malicious node A_2 starts broadcasting forged OGMs.

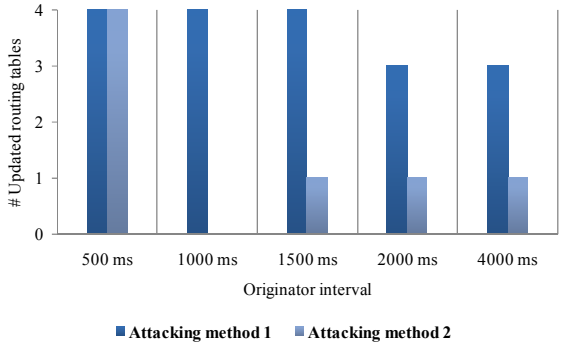
We used the two routing manipulation attack techniques introduced in section IV:

- **Method 1**. We generate OGMs with *sequence numbers* values at least twice the values of node O_1's *sequence numbers*, e.g., when we start executing the attack if the actual *sequence number* of node O_1 is 500 we generate OGMs with *sequence numbers* starting from 1000.
- **Method 2**. The generated OGMs have *sequence numbers* values close to the actual *sequence numbers* values of node O_1.

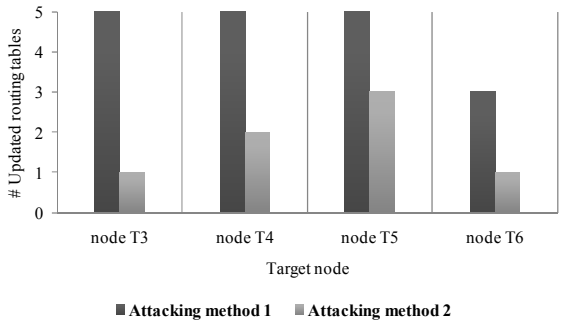
Furthermore, we take into account different *originator interval* values while broadcasting the phony OGMs to verify the attack efficacy. We used *originator interval* values of 500 ms, 1000 ms, 1500 ms, 2000 ms, and 4000 ms.

We saved the routing tables of the target nodes before staring the attack to verify Originator O_1's route entry. This route entry should point to a route via node T_3 because the packet loss rate of the link between nodes O_1 and T_3 is lower than the one between nodes O_1 and A. We also saved the target nodes' routing tables after each attack execution to check if node O_1's route entry was changed as the result of the phony OGMs broadcasted by the attacker.

Figure 5 shows the results obtained. For attacking method 1 and *originator interval* values of 500 ms, 1000 ms, and 1500 ms, we can easily notice that the number of OGMs generated by node A_2 win the neighbor ranking of all target nodes T_3, T_4, T_5, and T_6, as presented in Figure 5 (a). Accordingly, the target nodes update their routing tables, as presented in Figure 5 (b). Nevertheless, the routing table of node T_6 is not updated for *originator interval* values of 2000 ms and 4000 ms because the phony OGMs' broadcasting rate is lower than node O_1's default rate, i.e., the number of phony OGMs cannot override the number of legitimate OGMs. Therefore, node T_6 will not update its current best next-hop towards node O_1.



(a) Routing table updates per *originator interval*.



(b) Routing table updates per target node.

Figure 5. Routing table updates for scenario 1.

For attack method 2 and *originator interval* of 500 ms, we can notice all target nodes update their routing tables, as illustrated in Figure 5 (a). The broadcasting rate of the attacker is higher than the default rate of node O_1, so the phony OGMs continuously win the neighbor ranking. However, none of the routing tables is updated for *originator interval* of 1000 ms as the number of phony OGMs is not enough to surpass the number of legitimate OGMs. For the other *originator interval*, at last one routing table is updated because the

phony *sequence numbers* differ more than the *sliding window* size of node O_1.

The experiments results indicate that a compromised node can efficiently interfere in the mesh nodes' routing tables by broadcasting a sufficient number of fake OGMs for a legitimate mesh node. To make sure that the forged OGMs will always overcome the real OGMs the attacker can execute the attacking method 1 with *originator interval* values lower than 1500 ms, or attacking method 2 with *originator interval* values lower than 500 ms, i.e., the attacker must broadcast fake OGMs at a high rate.

5.2. Execution of the attack at scenario 2

The emulated mesh network applied for this scenario is illustrated in Figure 6. It comprises eleven nodes: two attackers A_4 and A_9, two gateway nodes O_1 and O_{10}, and seven target nodes T_2, T_3, T_5, T_6, T_7, T_8, and T_{11}. The attacker A_4 generates and broadcasts fake OGMs for node O_1, and attacker A_9 generates and broadcasts fake OGMs for node O_{10} to divert the routes of the target nodes toward nodes O_1 and O_{10} to nodes A_4 and A_9.

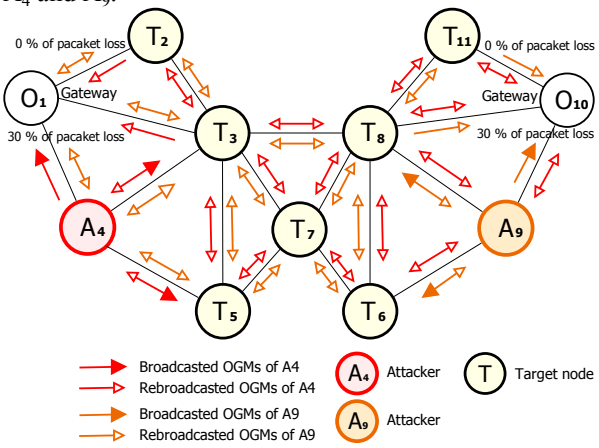
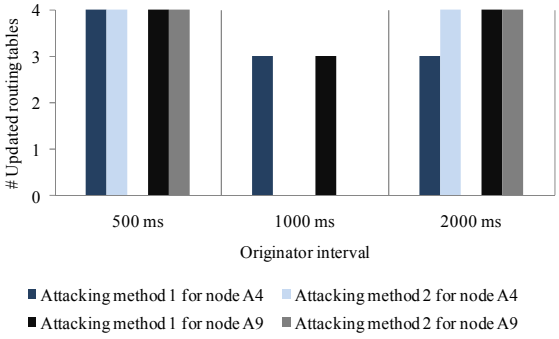


Figure 6. Mesh network for scenario 2.

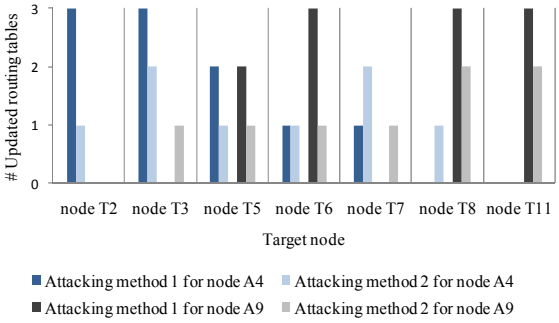
The duration of each experiment is 20 minutes. After 5 minutes of network initial set-up, the malicious nodes A_4 and A_9 start to generate the respective phony OGMs. In this scenario, we applied the two methods of routing manipulation attack introduced in section IV. To create and broadcast the fake OGMs for nodes O_1 and O_{10}, we executed the same strategy as in scenario 1. We consider *originator intervals* values of 500 ms, 1000 ms, and 2000 ms for both attacking methods.

Before executing the attack, we verify the route entries of nodes O_1 and O_{10} in routing tables of the target nodes. These routes entries should be pointing to paths via the nodes T_2 and T_{11} because these nodes have the best links to nodes O_1 and O_{10}, respectively. After executing the attacks, we check if the route entries of nodes O_1 and O_{10} have been altered in the routing tables of the victims as consequence of the fake OGMs broadcasted by the malicious nodes A_4 and A_9.

Figure 7 shows the results obtained. In Figure 7 (a), for attack method 1 and *originator interval* of 500 ms, 1000 ms, and 2000 ms, we can observe that most of target nodes update their routing tables because of attackers A_4 and A_9's OGMs. In Figure 7 (b), attacker A_4 forces the target nodes T_2, T_3, and T_5 to update their best ranking neighbor, and attacker A_9 has more influence on the routing tables of target nodes T_6, T_8, and T_{11}, since these target nodes are the only one-hop neighbors of the gateway nodes O_1 and O_{10} and the malicious nodes A_4 and A_9.



(a) Routing table updates per *originator interval*.



(b) Routing table updates per target node.

Figure 7. Routing table changes for scenario 2.

In Figure 7 (b), we can observe attacker A_4's fake OGMs had a limited influence on the nodes T_8 and T_{11}, and attacker A_9's fake OGMs did not convinced nodes T_2, T_3, and T_7 to update their routing table. As the fake OGMs of attackers A_4 and A_9 pass through the same best ranking neighbors, i.e., nodes T_3 and T_8, utilized by the OGMs of the gateway nodes O_1 and O_{10}, there is no reason for these target nodes to update their best ranking neighbor. For the same reason, the target node T_7 obtained the least number of routing table updates. On the contrary, the target nodes T_5 and T_6 are the only

nodes influenced by both attackers because they own more neighbors to select as best ranking neighbor.

In attack method 2 and using *originator interval* value of 500 ms, we can see that all of target nodes update their routing tables, as illustrated in Figure 7 (a). Since the number of fake OGMs is higher than the number of legitimate OGMs, the attackers continuously win the neighbor ranking as in method 1. None of the routing tables is updated for *originator interval* value of 1000 ms since the fake OGMs are not enough to predominate over the number of legitimate OGMs. The target nodes update their routing tables for *originator interval* value of 2000 ms because the *sequence numbers* sent by the attackers differ more than the *sliding window* size of target nodes.

From the experiments results, we can see that attackers can easily modify the routes of target nodes by employing attacking method 1 using *originator interval* values of 500 ms, 1000 ms, and 1500 ms, or employing attacking method 2 using *originator interval* values of 500 ms, and 1500 ms.

6. Detection of route manipulation attack

To detect the routing manipulation attack, we utilize BATMAN Advanced Control and Management tool (*batctl*) [17]. The *batctl* tool analyzes *batman-adv* debugging output, i.e., the log files, in order to create a small database of all *sequence numbers* broadcasted by the BATMAN nodes. Thus, the tool employs this database to trace all the *sequence numbers* of a given Originator by building a tree structure for every *sequence number*. This tree structure describes the paths a *sequence number* travel through the network when it is broadcasted by its Originator, then rebroadcasted by the neighboring nodes of the Originator, and until every Originator has receive it at least once. The tree's root node is the *sequence number*'s Originator, the root node's child nodes are the Originator's neighboring nodes, and so on.

For example, to determine whether a compromised node A is broadcasting forged *sequence numbers* for a legitimate node O_n. Firstly, we create the trees for node O_n's *sequence numbers* using *batctl* tool. After, we verify into the trees to find if any child node is equal to the root node, i.e., the own Originator O_n. If we see such mismatch, it means a legitimate OGM of node O_n was broadcasted back to node O_n, what is not possible according to BATMAN flooding mechanism, or a compromised node broadcasted an OGM for O_n and this OGM was rebroadcasted to node O_n, which is more reasonable.

We improved *batctl* tool to find this type of inconsistency when creating the trees for the Originator's *sequence numbers*. Thus, the tool was executed with the log files captured from the nodes of scenario 1 in Section 5.1 and the nodes of scenario 2 in Section 5.2.

6.1. Detecting the attack at scenario 1

The improved *batctl* tool traces all of the *sequence numbers* broadcasted by each Originator and counts the number of mismatches found for each node. Figure 8 presents the attack detection results for the mesh nodes of Figure 4 when *originator interval* is 500 ms.

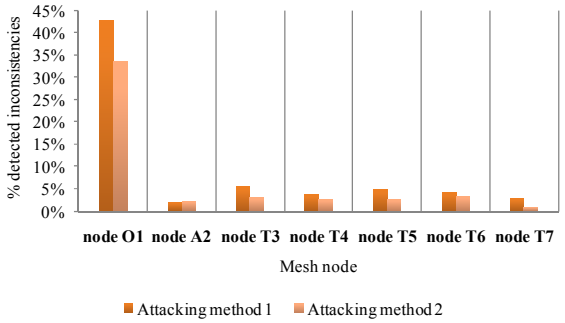


Figure 8. Detection results for scenario 1.

For both attacking methods 1 and 2, we can note Originator O_1 has the highest rate of inconsistencies detected in its *sequence number* trees, i.e., 43% and 33%. For attacking method 1, 43% of the OGMs broadcasted for node O_1 in the network are malicious, and for attack method 2, 33% of the OGMs broadcasted for node O_1 are malicious. Hence, we should verify the *sequence number* trees of node O_1 in order to find out who is broadcasting the fake *sequence numbers* for node O_1.

In Figure 8, for the other mesh nodes, the rate of detected inconsistencies is low; the values are below 6%. In fact, these inconsistencies do not correspond to attack attempts, and they can be considered as false positives. These inconsistencies occur at the beginning and at the end of each experiment, and they are caused by the lack of synchronization between the mesh nodes at the time the logging mechanism at each node starts and stops saving BATMAN debugging information. For instance, when node T_3 stops saving the debugging output, its log file contains more information about the broadcasted *sequence numbers* than the log files of nodes O_1 and A_2 that have stopped the capturing before. Therefore, the *batctl* tool is not able trace the complete path of certain *sequence numbers* in the log files, and then builds spurious *sequence numbers* trees that cause these false inconsistencies.

Figure 9 illustrates an example of an inconsistency

detected for Originator O_1. The picture shows a *sequence number* tree of a phony OGM. In Figure 9, the compromised node generates and sends a *sequence number* for node O_1, which is the root of the tree. Thus, this *sequence number* is broadcasted on the interface of nodes A_2, O_1, and T_4 that are the child nodes of the root. Hence, the tool finds the mismatch, as the child node "O_1" is equal to the root node "O_1".

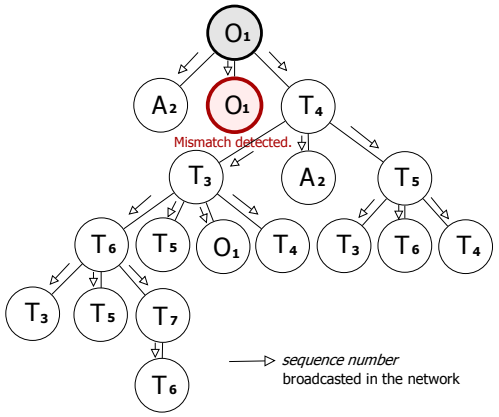


Figure 9. *Sequence number* tree for mesh node O_1.

Moreover, we can perceive that the attack source is node A_2 because the OGM is broadcasted to nodes A_2, O_1, and T_4 in the same time. In addition, according to the mesh topology of Figure 4, the only node A_2's one-hop neighbors are the nodes O_1, and T_4.

Figure 10 shows the detection results for node O_1 at different *originator interval* values.

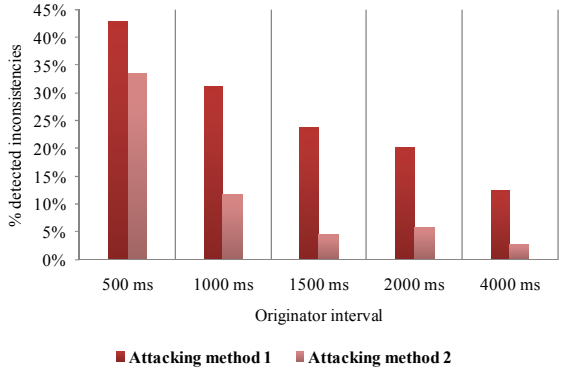


Figure 10. Detection results for node O_1.

We can observe that, for attacking methods 1 and 2, the rate of detected inconsistencies decreases as the *originator interval* value increases. The decrease in the rate of detected inconsistencies can be considered as normal since the broadcasting rate of fake *sequence numbers* also decreases accordingly, what means less fake OGMs being broadcasted.

Nonetheless, for attack method 2, the detection rate values for *originator intervals* of 1000 ms, 1500 ms, 2000 ms, and 4000 ms are even lower than the detection rate values of attack method 1. Since the generated *sequence numbers* values are close to the real *sequence numbers* values, the fake *sequence numbers* are overlapped by the real *sequence numbers* of node O_1. This prevents the tool to trace the real path of the fake *sequence numbers* in the network, so making it difficult for the tool to detect the mismatches.

The modified *batctl* tool had a good performance in the role of detecting fake OGMs broadcasted by the malicious node A_2 in the mesh network scenario of Figure 4. The tool was able to detect most of fake OGMs in attack method 1 and most of fake OGMs in attack method 2 when *originator interval* is 500 ms. In these cases the route manipulation attack is more effective in redirecting the routes of target nodes, as presented in Figure 5's results.

6.2. Detecting the attack at scenario 2

Figure 11 presents the detection results for the mesh nodes of Figure 6 when *originator interval* is 1000 ms.

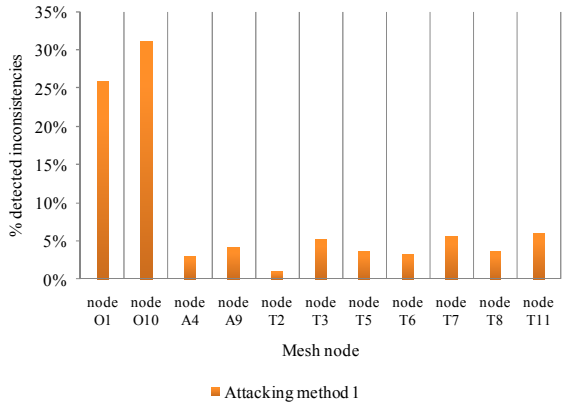


Figure 11. Detection results for scenario 2.

In this experiment, nodes O_1 and O_{10} had the highest rate of detected inconsistencies. We found that 26% of the *sequence numbers* broadcasted for Originator O_1 contains mismatches and 31% of the *sequence numbers* broadcasted for Originator O_{10} contains incompatibilities. These detection rate values are close to the detection rate value of Originator O_1 in scenario 1, as shown in Figure 10. For the other nodes, we can see that the rate of detected inconsistencies is below 6%. As explained on Section 6.1, these inconsistencies do not correspond to real attack attempts.

The modified *batctl* tool was able to detect the fake OGMs broadcasted by malicious node A_4 for node O_1 and the fake OGMs broadcasted by malicious node A_9

for node O_{10} in the network scenario of Figure 6. In this scenario, the route manipulation attack was well succeeded in redirecting the routes of target nodes, as showed in the results of Figure 7.

7. Conclusions

We have analyzed how an attacker can manipulate the routes of BATMAN mesh nodes. The attack consists of generating and broadcasting forged OGMs for a legitimate mesh node to redirect the routes of the mesh nodes to the malicious node.

The results of the experiments have shown that the routing operations of the mesh network can seriously affected by flooding forged OGMs. In addition, BATMAN protocol has proved to be vulnerable to route manipulation attack. However, this type of attack can be detected by tracing the OGMs broadcasted by each node in the network. To find the source of the attack we rely on the topology of the mesh network.

As future work, we plan to employ network intrusion detectors at the mesh nodes to monitor the traffic in real time and identify ongoing attacks at the same time.

10. References

[1] W. Zhang, Z. Wang, S. K. Das, and M. Hassan, "Security issues in wireless mesh networks", In Book Wireless Mesh Networks: Architectures and protocols. New York: Springer, 2008.

[2] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," Computer Networks, vol. 47, pp. 445-487, Jan. 2005.

[3] D. Johnson, N. Ntlatlapa, and C. Aichele, "A simple pragmatic approach to mesh routing using BATMAN", In IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries, October 2008.

[4] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. BeldingRoyer, "A Secure routing protocol for ad hoc networks," In Proceedings of 2002 IEEE International Conference on Network Protocols (ICNP), November 2002.

[5] Y-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: a secure on-demand routing protocol for ad hoc networks," in Proceedings of the MobiCom 2002, Atlanta, Georgia, USA, September 23-28, 2002.

[6] M. G. Zapata, "Secure ad hoc on-demand distance vector routing," ACM Mobile Computing and Communications Review (MC2R), Vol. 6, No. 3, pp. 106-107, July 2002.

[7] X. Wu, and N. Li, "Achieving privacy in mesh networks", In: SASN '06: proceedings of the fourth ACM workshop on security of ad hoc and sensor networks. ACM, New York, pp 13–22.

[8] T. Wu, Y. Xue, and Y. Cui, "Preserving traffic privacy in wireless mesh networks", In: WOWMOM '06: proceedings of the 2006 international symposium on world of wireless, mobile, and multimedia networks. IEEE Computer Society, Washington, DC, pp 459–461.

[9] L. Santhanam, D. Nandiraju, N. Nandiraju, and D. Agrawal, "Active cache based defense against dos attacks in wireless mesh network", In Wireless pervasive computing, 2007, ISWPC '07, 2nd international symposium, San Juan, 5–7, February 2007.

[10] Md. S. Islam, Md. A. Hamid, B. G. Choi, and C. S. Hong, 'Securing layer-2 path selection in wireless mesh networks', in 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, pp. 69-83.

[11] B. Kannhavong, H. Nakayama, N. Kato, Y. Nemoto, and A. Jamalipour, "A collusion attack against OLSR-based mobile ad hoc networks", In Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE, pages 1--5, November 2006.

[12] D. Raffo, C. Adjih, T. Clausen, and P. Muhlethaler, "Securing OLSR using node locations", Proc. 2005 Euro. Wireless, Nicosia, Cyprus, Apr. 10–13, 2005.

[13] T. Clausen, and P. Jacquet, "Optimized link state routing protocol (OLSR)", IETF RFC 3626 (Experimental), October 2003.

[14] D. B. Johnson, D. A. Maltz, and Y-C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (DSR)", IETF Internet Draft, draft-ietf-manet-dsr-09, April 2003.

[15] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) Routing," IETF RFC 3561, July 2003.

[16] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, "Better approach to mobile ad-hoc networking (B.A.T.M.A.N.)", April 2008, IETF Internet-Draft (expired October 2008), [Online], available at http://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00.

[17] Open-Mesh.net, "B.A.T.M.A.N. (better approach to mobile ad-hoc networking)", [Online], available at http://www.open-mesh.net/.

[18] QEMU, "machine emulator and virtualizer", [Online], available at http://wiki.qemu.org.

[19] VDE switch, "Virtual Distributed Ethernet switch", [Online], available at http://wiki.virtualsquare.org/wiki/index.php/VDE.

[20] PackETH, "Ethernet packet generator", [Online], available at http://packeth.sourceforge.net/.