How to Misuse AODV: A Case Study of Insider Attacks against Mobile Ad-hoc Routing Protocols

Peng Ning and Kun Sun
Computer Science Department, North Carolina State University
Raleigh, NC 27695-7534
Emails: ning@csc.ncsu.edu, ksun3@unity.ncsu.edu

February 23, 2003

Abstract

This paper presents a systematic analysis of insider attacks against mobile ad-hoc routing protocols, using the Ad hoc On-Demand Distance Vector (AODV) protocol as an example. It identifies a number of attack goals and then study how to achieve these goals through misuses of the routing messages. To facilitate the analysis, this paper classifies the insider attacks into two categories: *atomic misuses* and *compound misuses*. Atomic misuses are performed by manipulating a single routing message, which cannot be further divided; compound misuses are composed of combinations of atomic misuses and possibly normal uses of the routing protocol. The analysis results in this paper reveal several classes of insider attacks, including *route disruption, route invasion, node isolation,* and *resource consumption*. This paper also includes simulation results that demonstrate the impact of these attacks.

1 Introduction

Mobile Ad-hoc Networks (MANET) have attracted substantial research efforts recently, partially due to their attractive applications in infrastructureless situations such as battle fields and disaster recovery. Among all the research issues, security in MANET is particularly challenging due to the nature of wireless communication and the lack of infrastructure supports. Several efforts (e.g., Security-aware AODV [1], Ariadne [2], SEAD [3], CONFIDANT [4], watchdog and pathrater [5]) are under way to provide security services in ad-hoc routing protocols.

Most of the current security mechanisms (e.g., Ariadne [2], SEAD [3]) are preventive approaches that depend on cryptography to ensure the security of the network. However, in a typical mobile ad-hoc network such as a battle field, mobile nodes are extremely vulnerable to capture or key compromise. Even if critical keying materials are protected by tamper proof hardware, it is still difficult to ensure that the same hardware will not be misused by an attacker. Thus, to ensure the overall security of the network, it is important to develop security mechanisms that can survive malicious attacks from "insiders" who have full control of some nodes. In order to protect against insider attacks, it is necessary to understand how an insider can attack a wireless ad-hoc network. Several attacks (e.g., routing disruption attacks and resource consumption attacks [2, 3]) have been discussed in the literature. However, these attacks have not been seriously studied and verified.

In this paper, we adopt a systematic way to study the insider attacks against mobile ad-hoc routing protocols. We first identify a number of misuse goals that an inside attacker may want to achieve, and then enumerate all possible actions that an attacker may apply to a routing message. Our analysis is then to examine whether the attack goals may be achieved through these misuse actions. To facilitate the analysis, we further classify misuses of the AODV protocol into two categories: *atomic misuses* and *compound misuses*. Intuitively, atomic misuses are performed by manipulating a single routing message, which cannot be further divided. In contrast, compound misuses are composed of multiple atomic misuses, and possibly normal uses of the routing protocol.

Since atomic misuses are potentially building blocks of compound misuses, in this paper, we start with analyzing atomic misuses. In addition, we also study compound misuses that can achieve more powerful effects than simple compositions of atomic misuses when carefully composed together. We do not discuss simple compositions of atomic misuses, though we do perform simulation experiments to study their impacts. We pick the AODV protocol [6] as a target, performing our analysis from an attacker's perspective. It is easy to see that our analysis scheme is also applicable to other ad-hoc routing protocols, possibly with slight changes. To validate the analysis results, we have implemented the misuses based on the AODV extension in ns2, and evaluated the effectiveness of the misuses through simulations.

The rest of this paper is organized as follows. The next section briefly describes the AODV protocol. Section 3 describes our analysis scheme. Section 4 focuses on analyzing the atomic misuses of AODV routing message. Section 5 discusses compound misuses. Section 6 presents the experimental results. Section 7 discusses the related work in security of wireless ad-hoc networks. Section 8 concludes this paper and points out future research directions. Appendices give the details of the misuses as well as the simulation results.

2 An Overview of AODV Protocol

The Ad-hoc On-Demand Distance Vector (AODV) [6] protocol is an on-demand routing protocol, which initiates a route discovery process only when desired by a source node. When a source node wants to send data packets to a destination node but cannot find a route in its routing table, it broadcasts a Route Request (RREQ) message to its neighbors. Its neighbors then rebroadcast the RREQ message to their neighbors if they do not have a *fresh enough* route to the destination node. (A fresh enough route is a valid route entry for the destination node whose associated sequence number is equal to or greater than that contained in the RREQ message.) This process continues until the RREQ message reaches the destination node or an intermediate node that has a fresh enough route.

Every node has its own sequence number and RREQ ID[1]. AODV uses sequence numbers to guarantee that all routes are loop-free and contain the most recent routing information. RREQ ID in conjunction with source IP address uniquely identify a particular RREQ message. The destination node or an intermediate node only accepts the first copy of a RREQ message, and drops the duplicated copies of the same RREQ message.

After accepting a RREQ message, the destination or intermediate node updates its *reverse route* to the source node using the neighbor from which it receives the RREQ message. The reverse route will be used to send the corresponding Route Reply (RREP) message to the source node. Meanwhile, it updates the sequence number of the source node in its routing table to the maximum of the one in its routing table and the one in the RREQ message. When the source or an intermediate node receives a RREP message, it updates its forward route to the destination node using the neighbor from which it receives the RREP message. It also updates the sequence number

[1] It is also known as flood ID in earlier versions of AODV specifications.

of the destination node in its routing table to the maximum of the one in its routing table and the one in the RREP message. A Route Reply Acknowledgement (RREP-ACK) message is used to acknowledge receipt of a RREP message. Though not required, AODV may utilize the HELLO message to maintain the local connectivity of a node.

Route maintenance is done with Route Error (RERR) messages. If a node detects a link break in an active route, it sends out a RERR message to its upstream neighbors that use it as the next hop in the broken route. When a node receives a RERR message from its neighbor, it further forwards the RERR message to its upstream neighbors.

AODV is a stateless protocol; the source node or an intermediate node updates its routing table if it receives a RREP message, regardless of whether it has sent or forwarded a corresponding RREQ message before. If it cannot find the next hop in the reverse routing table, it simply drops the RREP message. Otherwise, it unicasts the RREP message to the next hop in the reverse route.

In general, a node may update the sequence numbers in its routing table whenever it receives RREQ, RREP, RERR, or RREP-ACK messages from its neighbors.

3 Analysis Scheme

We adopt a systemic way to analyze the insider attacks against the AODV protocol. We first identify a number of misuse goals that an inside attacker may want to achieve, and then study how these goals may be achieved through misuses of the routing messages. These misuse goals are listed as follows.

- *Route Disruption (RD).* Route Disruption means either breaking down an existing route or preventing a new route from being established.
- *Route Invasion (RI).* Route invasion means that an inside attacker adds itself into a route between two endpoints of a communication channel.
- *Node Isolation (NI).* Node isolation refers to preventing a given node from communicating with any other node in the network. It differs from Route Disruption in that Route Disruption is targeting at a route with two given endpoints, while node isolation is aiming at all possible routes.
- *Resource Consumption (RC).* Resource consumption refers to consuming the communication bandwidth in the network or storage space at individual nodes. For example, an inside attacker may consume the network bandwidth by either forming a loop in the network.

There may be other attack goals (e.g., denial of service); however, we do not consider them in our current work.

To facilitate the analysis, we further classify misuses of the AODV protocol into two categories: *atomic misuses* and *compound misuses*. Intuitively, atomic misuses are performed by manipulating a single routing message, which cannot be further divided. In contrast, compound misuses are composed of multiple atomic misuses, and possibly normal uses of the routing protocol. It is easy to see that atomic misuses may be used as building blocks of compound misuses.

We perform our analysis of atomic misuses through understanding the effects of possible *atomic misuse actions*. Each atomic misuse action is an indivisible manipulation of one routing message. Specifically, we divide the atomic misuse actions in AODV into the following four categories:

- *Drop (DR).* The attacker simply drops the received routing message.
- *Modify and Forward (MF).* After receiving a routing message, the attacker modifies one or several fields in the message and then forwards the message to its neighbor(s) (via unicast or broadcast).
- *Forge Reply (FR).* The attacker sends a faked message in response to the received routing message. Forge Reply is mainly related to the misuse of RREP messages, which are in response of RREQ messages.
- *Active Forge (AF).* The attacker sends a faked routing message without receiving any related message.

At first glance, compound misuses seem to be simple compositions of atomic uses. However, when carefully aggregated together, some compositions of atomic misuses become more powerful attacks due to the changes in the number of messages. For example, if an attacker regularly broadcasts RREQ messages with false information in the neighborhood of a victim node, the attacker can successfully prevent the victim node from receiving any messages. In our analysis of compound misuses, we focus on the aforementioned, "powerful" compound misuses; simple compositions of atomic misuses (and possibly the normal routing messages as well as the above compound misuses) can be analyzed via automatic vulnerability analysis tools (e.g., attack graphs [8]).

It is easy to see that our analysis scheme is also applicable to other mobile ad hoc routing protocols, possibly with slight modification. However, in this paper, we only focus on the AODV protocol, while considering the analysis of the other protocols as possible future work.

Since atomic misuses form the foundation of compound misuses, in the following, we first perform a systematic analysis of atomic misuses of the AODV protocol, and then study how atomic misuses and normal routing messages may be combined to launch compound misuses.

4 Atomic Misuses of AODV

In this section, we present our analysis results about atomic misuses of the AODV protocol. Due to the space limit, we only summarize the results and discuss a few atomic misuses in details. For the complete set of atomic misuses, please refer to the appendix of our technical report [7].

In our analysis, we use a simple naming scheme to identify atomic misuses, which combines routing message type and atomic misuse action. Specifically, each atomic misuse is named in the form of `MessageType_Action`, which means that an inside attacker applies the "`Action`" to a routing message of type "`MessageType`." For brevity, we use the abbreviations introduced in the previous section to represent atomic misuse actions. For example, `RREP_DR` represents that an attacker drops (DR) a RREP message. We also use names in the form of `MessageType_Action_Goal` to represent that an inside attacker attempts to achieve the "`Goal`" by applying the "`Action`" to a routing message of type "`MessageType`." For example, `RREP_DR_RD` represents that an attacker attempts to disrupt (RD) a route by dropping (DR) a RREP message.

4.1 Atomic Misuses of RREQ Messages

Table 1 summarizes the atomic misuses of a RREQ message. The atomic misuse action *Forge Reply* is not applicable to RREQ messages, since RREQ messages are not used to reply to any other routing message.

Table 1: Atomic Misuses of A RREQ Message and Achievable Misuse Goals.

Atomic Misuse	Route Disruption	Route Invasion	Node Isolation	Resource Consumption
`RREQ_DR`	Yes (in some cases)	No	No	No
`RREQ_MF`	Yes	Yes	Partial[2]	No
`RREQ_AF`	Yes	Yes	Partial	No

Atomic misuse `RREQ_DR` refers to simply dropping the received RREQ message. If an attacker applies such attacks to all the RREQ messages it receives, this kind of misuses is equivalent to not having the attacking node in the network. An inside attacker may also selectively drop RREQ messages. Attackers that launch such misuses are in nature similar to the selfish nodes mentioned in [5].

Atomic misuse `RREQ_MF` refers to the atomic misuses with which an inside attacker modifies one or several fields in a RREQ message that it just receives, and then broadcasts the modified RREQ message. Table 2 lists the RREQ message fields that an attacker may modify as well as the possible modifications.

Table 2: Possible Modifications of Fields in A RREQ Message.

RREQ Message Field	Modifications
Type	Change the message type.
RREQ ID	Increase it to make the faked RREQ message acceptable, or decrease it to make the RREQ message unacceptable.
Hop Count	Decrease it to update other nodes' reverse routing tables, or increase it to invalidate the update.
Destination IP Address	Replace it with another IP address.
Destination Sequence Number	Increase it to update other nodes' forward route tables, or decrease it to suppress its update.
Source IP Address	Replace it with another IP address.
Source Sequence Number	Increase it to update other nodes' reverse route tables, or decrease it to suppress its update.
Flags	Reverse the setting.

Several fields have immediate security implications when modified. RREQ ID along with the source IP address uniquely identifies a RREQ message; they indicate the freshness of a RREQ message. Since a node only accepts the first copy of a RREQ message, an increased RREQ ID along with the source IP address can guarantee that the faked RREQ message is accepted by other nodes.

To ensure loop freedom in AODV, after receiving a RREQ message, a node updates its reverse routing table only if the source sequence number field in the RREQ message is greater than that in its routing table, or the source sequence numbers are equal, but the hop count field in the RREQ message is smaller than that in the routing table. An inside attacker may also change these fields to affect other nodes' routing table.

An intermediate node or a source node updates its forward routing table if the destination sequence number in the RREP message is greater than the one in its routing table, or the destination sequence numbers are the same, but the hop count in the RREP message plus one is smaller than the one in its routing table. An inside attacker may increase the sequence numbers or decrease the hop count in a faked RREQ message to update other nodes' routing tables, or decrease the sequence numbers or increase the hop count to invalidate a RREQ message.

When a node updates its routing table, the next hop in the route entry is assigned as the node from which it receives the RREQ message. An inside attacker may manipulate the source IP address in the IP header to change the reverse route.

Both `RREQ_DR` and `RREQ_MF` must be triggered by an incoming RREQ message. In contrast, an inside attacker may perform a `RREQ_AF` misuse to forge a RREQ message without receiving a RREQ message. An inside attacker may need to collect some necessary information to forge RREQ messages (e.g., by listening to the traffic). Theoretically, the attacker may forge any field in a RREQ message, generating the effects we just discussed.

Now let us look at an atomic misuse of a RREQ message, `RREQ_MF_NI`, with which an inside attacker prevents a victim node from receiving data packets from other nodes for a short period of time. The attacker may make the following modifications after it receives a RREQ message from the victim node: (1) Increase the RREQ ID by a small number; (2) Replace the destination IP address with a non-existent IP address; (3) Increase the source sequence number by at least one; (4) Set the source IP address in IP header to a non-existent IP address. The attacker then broadcasts the forged message. When the neighbors of the attacker receive the faked RREQ message, they will update the next hop to the source node to the non-existent node, since the faked RREQ message has a greater source sequence number. Due to the non-existent destination IP address, the faked message can be broadcasted to the farthest nodes in the ad-hoc network. When other nodes want to send data packets to the source node, they will use the routes established by the faked RREQ message, and the data packets will be dropped due to the non-existent node.

This atomic misuse can prevent a victim node from receiving data packets for a short period of time; however, it cannot fully isolate the victim node, due to the local repair mechanism in the AODV protocol [6]. The other nodes will initiate another round of route discovery if they note that the data packets cannot be delivered successfully. In addition, the victim node may still be able to send data packets to other nodes.

Several of the atomic misuses of RREQ messages use RREQ messages to add entries the routing table of other nodes. These entries are different from those established through normal exchange of RREQ and RREP messages. In particular, the lifetime of these entries is set to a default value (e.g., 3 seconds as in our experiments). Thus, to make such entries effective, an attacker needs to launch the atomic misuses periodically.

4.2 Atomic Misuses of RREP Messages

Table 3 summarizes the atomic misuses of a RREP message and whether they can achieve the misuse goals. The premise of atomic misuses of RREP messages is that the inside attacker must

Table 3: Atomic Misuses of A RREP Message and Achievable Misuse Goals.

Atomic Misuse	Route Disruption	Route Invasion	Node Isolation	Resource Consumption
`RREP_DR`	Yes (in some cases)	No	No	No
`RREP_MF`	Yes	Yes	No	No
`RREP_FR`	Yes	Yes	No	No
`RREP_AF`	Yes	Yes	No	Yes

already be in a reverse route involving a victim node, so that it can receive a RREQ or RREP message, or send a forged RREP through some other nodes. Due to this restriction, most of the atomic misuses of RREP messages, including `RREP_DR` `RREP_MF`, have limited impact.

Atomic misuse `RREP_FR` is specific to RREP messages. It refers to the misuse with which an

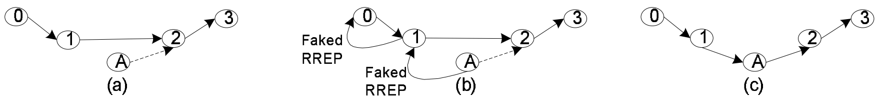


Figure 1: An Attacker Invades A Route by Sending A Faked RREP Actively.

attacker forges a RREP message in response to a RREQ message. For example, after receiving a RREQ message, an inside attacker may forge a RREP message as if it had a fresh enough route to the destination node. In order to suppress other legitimate RREP messages that the source node may receive from other nodes, the attacker may forge a faked RREP message by increasing the destination sequence number. An attacker may disrupt the route between the victim node to a given destination, or invade in the route between by suppressing other alternative routes.

An interesting atomic misuse is `RREP_AF_RI`. If an inside attacker has routes to both the source and the destination nodes of an existing route (as shown in Figure 1(a)), it can invade the route by sending a faked RREP message to the source node. In Figure 1, assume node A is the attacking node, which already has a route to nodes 0 and 3, respectively. Node A can forge a RREP message as follows: (1) Set the source IP to node 0; (2) Set the destination IP to node 3; (3) Set the destination sequence number to node 3's sequence number plus at least one; (4) Set the source IP in the IP header to node 2; (5) Set the destination IP in the IP header to node 1. Node A then sends the faked RREP message to node 1, which forwards the faked RREP message to node 0 (Figure 1(b)). When nodes 0 and 1 receive the faked RREP message, they will update the sequence number of node 3 in their routing tables to the destination sequence number in the faked RREP message. Node 0 will still use node 1 as the next hop to node 3, but node 1 will update node A as the next hop to node 3. Note that node A already has a route to node 3. As a result, node A successfully becomes a part of the route from node 0 to node 3 (Figure 1(c)).

4.3 Atomic Misuses of RERR Messages

Table 4 summarizes the three types of atomic misuses of RERR messages and the misuse goals that they can achieve. The misuse action *Forge Reply* is not applicable to RERR messages, since RERR messages are not used to reply to any routing messages.

Table 4: Atomic Misuses of A RERR Message and Achievable Misuse Goals.

Atomic Misuse	Route Disruption	Route Invasion	Node Isolation	Resource Consumption
`RERR_DR`	Yes (in some case)	No	No	No
`RERR_MF`	Yes	No	Partial	Yes
`RERR_AF`	Yes	No	Partial	Yes

`RERR_DR` has limited impact on the network except for causing delays in the identification of route errors, since the upstream nodes will eventually discover the problematic routes and establish new routes.

In order to know which neighbors should receive a RERR message, each node keeps a "precursor list" of its neighbors for each route entry. When a link break is detected, the node sends a RERR message to all the nodes in the corresponding precursor list. To launch `RERR_MF` misuses, an inside attacker may modify the RERR message after it receives a RERR message, and send the faked RERR message to the neighbors in the precursor list. Table 5 lists the fields in a RERR message that the attacker may manipulate. Sometimes, the attacker may modify the IP addresses in the IP header as well.

Table 5: Possible Modifications of Fields in A RERR message.

RERR Message Field	Modifications
Type	Change the value of Type.
DestCount	Modify it according to the number of unreachable destinations included in the RERR message.
Unreachable Destination IP Address	Replaces it with another IP address.
Unreachable Destination Sequence Number	Increases it to update other nodes' routing table, or decreases it to invalidate this entry.
Additional Unreachable Destination IP address (if needed)	Add a new destination IP address which is still reachable.
Additional Unreachable Destination Sequence number (if needed)	Increases it to update other nodes' routing table, or decreases it to invalidate this entry.

5 Compound Misuses

One or several inside attackers may combine atomic misuses, and possibly normal uses of routing messages, in any order to launch compound misuses. For example, an attacker may repeatedly launch the same type of atomic misuses to make the impact persistent. As another example, an attacker may launch some early atomic or compound misuses to prepare for some later ones. A crucial issue here is to understand the compound misuses that can be used as "building blocks" of more complex attacks. Once we understand "building blocks," we may analyze the complex atomic scenarios through automatic vulnerability analysis tools such as the attack groups [8].

For convenience, we extend the naming scheme for atomic misuses to denote compound misuses of the same type of atomic misuses. Specifically, we put an "s" after the type of routing message that is being misused in the corresponding atomic misuse. For example, `RREQs_AF` represent that an attacker actively forges multiple RREQ messages.

In our analysis, we observe that most of atomic misuses targeted at disrupting services can only generate temporary impact due to the local repair mechanism that is commonly seen in mobile ad hoc routing protocols. Thus, to make the impact of these misuses persistent, an attacker needs to repeat the atomic misuses regularly. We do not discuss such misuses in detail; however, our experimental results will show that an attacker can indeed achieve its goals through such compound misuses.

Another class of compound misuses is more interesting than simply repeating the same type of atomic misuses. We discovered that an attacker may achieve some misuse goals through well planned combination of atomic misuses. Let's see an example as follows.

An inside attacker may invade into a route through a `RREQs_AF` compound misuse. Consider the scenario shown in Figure 2(a). Suppose nodes 0 through 5 are normal nodes, and node A is the attacker node. Further assume there is a route from node 0 to node 5. The attacker at node A may forge a RREQ message as follows: (1) Set the source IP address as node 5; (2) Set the destination IP address as node 0; (3) Set the source sequence number to a number greater than node 5's current sequence number; (4) Set the source IP address in IP header as node A. Node A then broadcasts the faked RREQ message. After receiving this message, nodes 2 and 3 will both set node A as the next hop to node 5, as in Figure 2(b).

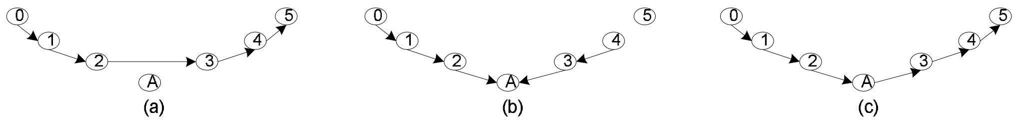


Figure 2: Route Invasion by Two Faked RREQ Messages.

To further establish the route from node A to node 5, the attacker may generate the second RREQ message as follows: (1) Set the source IP address as node A; (2) Set the destination IP address as node 5; (3) Set the destination sequence number to a number greater than node 5's current sequence number; (4) Set the source IP address in the IP header as node A. Node A can then broadcast this RREQ message. This message will help node A establish a route to node 5, as in Figure 2(c).

As discussed earlier, one or several inside attackers may compose attacks by arbitrarily combining atomic and/or compound misuses. In particular, the attackers may use different misuses to complement each other. For example, `RREQs_AF` is effective in preventing a victim node from receiving messages from other nodes, and `RREP_AF` is effective in preventing other nodes from receiving from the victim node. By combining them together, the attacker(s) may successfully isolate a node. In addition, one or several inside attackers may use some misuses or normal routing messages to prepare for later misuses. For example, all RREP related misuses require a route involving both the attacker and the victim node. To prepare for such misuses, an attacker may use a normal RREQ message or an atomic misuse (e.g., `RREQ_AF`) to establish the required route. These misuses are interesting; however, we do not consider them in this paper. Indeed, manually analyzing such attacks is not the best option due to the potentially large search space for possible complex attack scenarios. A better solution is to model the individual misuses and then construct attack strategies through automatic tools. Our work in this paper provides the foundation required by such tools.

6 Experimental Results

In order to validate our analysis results, we have implemented all the misuses and performed a series of experiments through simulation. The simulation is based on ns2 version 9[3] with the CMU Monarch extension for the AODV protocol[4]. To take advantage of the existing AODV code, we implemented the atomic misuses by simply overriding the AODV agent's receive and send functions. Compound misuses are performed by repeating/combining the atomic misuses.

Table 6 shows the parameters used in our experiments. We used continuous bit rate (CBR) in all our experiments. In each simulation scenario, there are 5 mobile nodes if it is for atomic misuses, and 20 nodes if it is for compound misuses. In all the experiments, there is only one inside attacker in the ad hoc network. The field configuration is 1000 m $\times$ 600 m. The simulation runs for 100 simulated seconds. After arriving at a location, a node stays there for 2.0 seconds before moving to the next location. A source node sends 4 data packets per simulated seconds. There are at most 20 connections during each simulation run. In a node's transmission range (250m), other nodes can receive signals from this node directly. The physical link bandwidth is 2 Mbps.

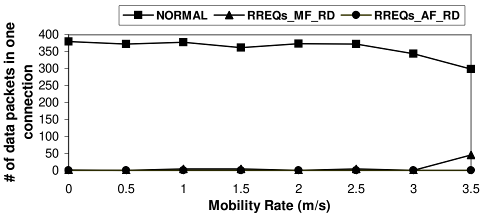
We have verified all the atomic misuses through analyzing the trace files generated by the sim-

[3] `http://www.isi.edu/nsnam/ns/`.

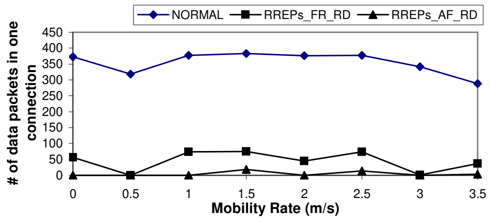
[4] `http://www.monarch.cs.rice.edu/`.

Table 6: Simulation Parameters

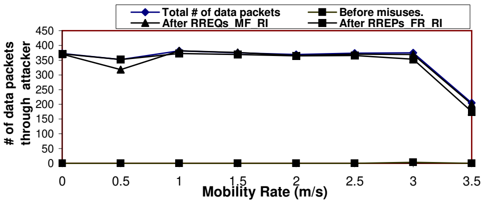
Communication Type	CBR
Number of Nodes	5 or 20
Simulation Area	1000m*600m
Simulation Time	100 seconds
Pause Time	2.0 seconds
Packet Rate	4 pkt/sec
Number of Connections	20
Transmission Range	250m
Physical Link Bandwidth	2Mbps
Number of Inside Attackers	1



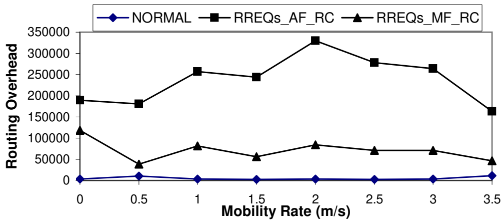
(a) Route Disruption by RREQs



(b) Route Disruption by RREPs



(c) `RREPs_FR_RI` and `RREQs_MF_RI`



(d) Resource Consumption by RREQs

Figure 3: Experimental Results about Compound Misuses

ulations. We found that all the atomic misuses intended for *Route Disruption* and *Node Isolation* succeeded; however, the effect can last for a short period of time due to the local repair mechanism in the AODV protocol. This is due to two reasons. First, the impact caused by such atomic misuses are detectable by the normal nodes, which then attempt to recover from the failures by establishing new routes. Second, all the atomic misuses are performed with a single routing message. They do not have further impact once the affected nodes perform local repair successfully.

In contrast, the atomic misuses intended for *Route Invasion* are much more subtle. Unless the routes established via atomic misuses are disrupted, the victim nodes will continue to use the routes involving the inside attacker to transmit data packets. Details of the experiments about atomic misuses can be found in the appendix.

Though atomic misuses for *Route Disruption* and *Node Isolation* are not effective when they are used individually, our experiments show that they are quite powerful when they are used in compound misuses.

Figure 3 shows the experimental results for compound misuses for *Route Disruption, Route Invasion*, and *Resource Consumption*. Figure 3(a) displays the numbers of data packets transmitted between two victim nodes when using compound misuses of RREQ messages. It clearly shows that when `RREQs_MF_RD` and `RREQs_AF_RD` are used against these two nodes, the number of data packets drops almost to zero. Figure 3(b) shows the same measure when compound misuses of RREP messages are used. The number of data packets transmitted between the two victim nodes is slightly better than in Figure 3(a); however, it is still much lower than the number of packets transmitted in normal situations. Figure 3(c) shows the number of data packets transmitted through an inside attacker with or without *Route Invasion* misuses. It is easy to see that the misuses effectively make the attacker a part of the route between the two victim nodes. Finally, Figure 3(d) shows that the routing overhead with `RREQs_MF_RC` is higher than the overhead in normal situations, and `RREQs_AF_RC` misuse is much higher than the `RREQs_MF_RC`.

7 Related Work

Research in MANET has been rather active. Several routing protocols have been proposed to discover and maintain routes in MANET environments, including secure routing protocols. Early proposals for secure ad hoc routing (e.g., [9], [1], [10], and [11]) use public key cryptography to protect ad hoc routing messages. However, due to the heavy computation involved in public key cryptography, these proposals are too expensive for nodes in mobile ad hoc networks, which are usually powered by batteries.

Recent results usually use symmetric cryptography to authenticate the routing messages. Papadimitratos and Haas proposed to authenticate the route discovery process with a secret key shared between the source and the destination nodes [12]. Basagni et al. use a network-wide secret key to secure the routing messages [13]. Yi et al. modified AODV to include security metrics for route discovery, using different trust levels with a shared symmetric key for each level [1]. Hu, Perrig, and Johnson have proposed a sequence of secure mobile ad hoc routing protocols, including Ariadne [2] and SEAD [3], as well as security mechanisms for routing protocols [14]. Their techniques include authenticating routing messages through a one-way key chain with delayed disclosures of keys, and authentication code with secret keys shared by mobile nodes.

Intrusion detection can provide another layer of protection to mobile ad hoc networks. Zhang and Lee proposed a distributed and cooperative IDS architecture in mobile ad-hoc networks [15]. They use data on the node's physical movements and the corresponding change in its routing table as the trace data to build the anomaly detection model. In Marti et al.'s proposal [5], each node uses a component called *watchdog* to detect misbehaving nodes, and another component called *pathrater* to choose a reliable route based on the information collected by the watchdog. In Buchegger and Boudec's proposal [4], each node not only monitors the bad behaviors of neighbors, but collects the list of malicious nodes from warnings sent from other trusted nodes.

8 Conclusions

In this paper, we reported the results of a systematic analysis of insider attacks against the AODV protocol. We classified the possible insider attacks into atomic misuses and compound misuses, and identified a number of atomic as well as compound misuses. We also performed a series of experiments (based on simulation) to validate these misuses. Our results showed that an inside attacker can effectively invade into routes or disrupt the normal operations of the AODV protocol.

The results in this paper represent our initial attempt in understanding insider attacks against mobile ad hoc routing protocols. As a part of our future work, we plan to investigate insider attacks against secure mobile ad hoc routing protocols such as Ariadne [2].

References

[1] S. Yi, P. Naldurg, and R. Kravets, "Security-aware routing protocol for wireless ad hoc networks," in *Proc. of ACM MobiHoc 2001*, Oct 2001.

[2] Y. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *Proc. of (MobiCom 2002)*, Sept. 2002.

[3] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," in *4th IEEE Workshop on Mobile Computing Systems and Applications*, June 2002.

[4] S. Buchegger and J. L. Boudec, "Performance analysis of the CONFIDANT protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks)," in *Proc. of ACM MobiHoc 2002*, pp. 226–236, June 2002.

[5] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proc. MobiCom 2000*, pp. 255–265, 2000.

[6] C. Perkins, E. Belding-Royer, and S. Das. Internet Draft, June 2002. draft-ietf-manet-aodv-11.txt.

[7] P. Ning and K. Sun, "How to misuse AODV: A case study of insider attacks against mobile ad-hoc routing protocols," Tech. Rep. TR-2003-07, CS Department, NC State University, 2003.

[8] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proc. of IEEE Symposium on Security and Privacy*, May 2002.

[9] L. Zhou and Z. J. Haas, "Securing ad hoc networks," *IEEE Network*, vol. 13, no. 6, pp. 24–30, 1999.

[10] J. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad hoc networks," in *Proc. ACM MobiHoc 2001*, 2001.

[11] K.Sanzgiri, B.Dahill, B.N.Levine, C.Shields, and E.M.Belding-Royer, "A secure routing protocol for ad hoc networks," in *Proc. of the Tenth IEEE Int'l Conf. on Network Protocols*, 2002.

[12] P. Papadimitratos and Z. J. Haas, "Secure routing for mobile ad hoc networks," in *Proc. of the SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, pp. 27 – 31, Jan 2002.

[13] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti, "Secure pebblenets," in *Proc. of ACM MobiHoc 2001*, pp. 156–163, 2001.

[14] Y. Hu, A. Perrig, and D. V. Johnson, "Efficient security mechanisms for routing protocols," in *Proc. of the 10th Annual Network and Distributed System Security Symposium*, 2003.

[15] Y. Zhang and W. Lee, "Intrusion detection in wireless ad hoc networks," in *Proc. of ACM MobiCom 2000)*, pp. 275–283, 2000.

A Description of Atomic Misuses and the Simulation Results

This appendix provides descriptions of the atomic misuses and the simulation results about these atomic misuses. In all the simulations, there are 5 nodes in the network, and node 2 is the malicious node launching the misuses.

A.1 Atomic Misuses of RREQ Messages

A.1.1 Atomic Misuses `RREQ_DR`

Atomic misuses `RREQ_DR` refer to simply dropping the received RREQ message. If an attacker applies such attacks to all the RREQ messages it receives, this kind of misuses is equivalent to not having the attacking node in the network. An inside attacker may also selectively drop RREQ messages. Such misuses are in nature similar to selfish nodes mentioned in [5]. If the attacking node is the only node between two parts of an ad-hoc network, it may selectively separate the nodes in these two parts, and partially achieve the goal of route disruption. `RREQ_DR` cannot achieve the other three misuse goals. Due to the simplicity of `RREQ_DR` misuses, we do not include the experimental results here.

A.1.2 Atomic Misuses `RREQ_MF`

`RREQ_MF_RD`: If an attacking node is the only node connecting two parts of an ad-hoc network, the attacker can prevent a new route from being established by utilizing one of the following modifications on a RREQ message it receives:

- Change the Type field;
- Replace the destination IP address with another IP address;
- Replace the source IP address with another IP address;
- Replace the source IP address in IP header with another IP address.

Even if there exists other routes between two given nodes, the attacker still has a chance to disrupt the new route from being established. Suppose node S broadcasts a RREQ message to establish a route to node D. After receiving the RREQ message, the attacker modifies the following fields of the RREQ message:

1. Replace the RREQ ID of node S with the RREQ ID of node D, and increases it by a small number;
2. Interchange the source IP address [5] (node S) with the destination IP address (node D) in the RREQ message;

[5]In draft-11, it is marked as "Originator IP Address".

3. Increment the destination sequence number by at least one, and then interchanges the source sequence number with the destination sequence number;

4. Fill source IP address in IP header with a non-existent IP address.

By doing these modifications, the attacker pretends to forward a RREQ message initiated from node D to node S, whereas the original RREQ message is initiated from node S to node D. Neighbors of the attacker accept the faked RREQ message since they have not received a RREQ message with such a RREQ ID from node D before. Because the faked RREQ message has a greater source sequence number, these neighbors updates their next hop to the node D as the non-existent node, which is indicated by the source IP address in the IP header. These neighbors rebroadcast the faked RREQ message to their neighbors. When node D receives the faked RREQ message, it just drops the message since it notices that this message is originated from itself. When node S receives the faked RREQ message, it updates its reverse route table since the source sequence number (of node D) in the faked RREQ message is greater than that in its route table. Node S then updates the next hop to node D as the neighbor from which it receives the faked RREQ message, and unicasts a RREP message to this neighbor. When the RREP message is unicasted along the reverse route, it is lost due to the non-existent node in the reverse route.

Due to the broadcast of the legitimate RREQ message, node S may receive normal RREP messages, but the route established by the faked RREQ message suppresses the routes established by RREP messages since node D's sequence number in the faked RREQ message is greater than those in the RREP messages. After that, node S begins to send data packets along the route established by the faked RREQ message, but all data packets are dropped when they reach the non-existent node. When the upstream neighbor of the attacker discovers the link failure, it either sends a RERR message back to node S, or starts "local repair," which broadcasts a RREQ message to discover a route from itself to the destination node if the destination is no farther than maximum repair hops away.

Note that the reverse route table (established by RREQ messages) and the forward route (established by RREP messages) are in indeed one route table. The route entries added by RREQ messages can be updated by RREP message, and vice versa.

In the simulation, node 0 is the source node, and node 1 is the destination node. Malicious node 2 sends a faked RREQ message to disrupt the route from node 0 to node 1. From the trace file, we can see clearly that the route is disrupted when node 3 attempts to forward data packets to node 88, which is a non-existent node. The fragment of the trace file is as follows:

...

s 0.200000000 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)

r 0.200900287 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)

s 0.201042333 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)

r 0.201942620 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)

s 0.201942934 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [88:255 -1:255 28 0] [0x2 3 101 [0 12] [1 10]] (REQUEST)

r 0.201943000 _4_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)

r 0.202843535 _4_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 28 0] [0x2 3 101 [0 12] [1 10]] (REQUEST)

r 0.202843535 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 28 0] [0x2 3 101 [0 12] [1 10]] (REQUEST)

...

s 0.778799135 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0

r 0.778799135 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0

s 0.778799135 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [3] 0 0

r 0.780535995 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [3] 1 0

f 0.780535995 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 88] [3] 1 0
D 0.780535995 _3_ RTR CBK 2 cbr 68 [13a 3 3 800] ——- [0:0 1:0 29 88] [2] 1 0
D 0.780535995 _3_ RTR CBK 3 cbr 68 [13a 3 3 800] ——- [0:0 1:0 29 88] [3] 1 0
s 0.780535995 _3_ RTR — 0 AODV 32 [0 0 0 0] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 0.781932282 _0_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 0.781932596 _2_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 0.781932662 _4_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
...

`RREQ_MF_RI`: Let's first consider a scenario in which an inside attacker is in the transmission range of a source node that initiates a RREQ message. After receiving the RREQ message from the source node, the attacker may modify the RREQ message as follows:

1. Increase the source node's RREQ ID by at least one;
2. Increase the source sequence number by at least one;
3. Increase the destination sequence number by at least one.

After generating this faked RREQ message, the attacker broadcasts it to its neighbors. The neighbors of the attacker accept the faked RREQ message due to the new pair of the RREQ ID and the source IP address. They then update their next hop to the source node as the attacking node, because the faked RREQ message has a greater source sequence number than those in their route tables. They also rebroadcast the faked RREQ message to their neighbors. When the source node receives the faked RREQ message, it drops the message, since this message appears to originate from itself. When the destination node receives the faked RREQ message, it updates its next hop to the source node as the neighbor from which it receives the faked RREQ message, and then updates its own sequence number to the maximum of its current sequence number and the destination sequence number in the RREQ message. After that, it fills the updated sequence number into the destination sequence number in the RREP message. The destination node then unicasts the RREP message to the source node along the reverse route, which includes the attacker. Because this RREP message contains a greater destination sequence number than that in the source node's route table which may have been updated by other legitimated RREP messages, the source node updates the destination sequence number to that in the RREP message, and sets the attacker as the next hop to the destination node. Now the attacker succeeds in invading the route from the source node to the destination node.

When the attacker is not in the transmission range of a source node, i.e., there exists at least one intermediate node between the attacker and the source node, the attacker cannot invade the route by modifying a RREQ message in the above way. When the attacker broadcasts the faked RREQ message, all the neighbors will accept it and update the attacker as the next hop to the source node. When the attacker forwards the RREP message to a neighbor along the reverse route, this neighbor will just send the RREP message back to the attacker. As a result, there will be a loop involving the attacker and one of its neighbors. However, the attacker still can invade the route by sending two faked RREQ messages, which is a compound misuse by `RREQs_MF_RI`.

In the simulation, node 0 is the source node, and node 1 is the destination node. In normal situations, the route is $0 \Rightarrow 3 \Rightarrow 1$; with atomic misuse `RREQ_MF_RI`, the route is $0 \Rightarrow 3 \Rightarrow 2 \Rightarrow 1$. The fragment of the trace file is as follows:

...
s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0

```
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519249 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.670400475 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [2:255 -1:255 29 0] [0x2 2 3 [1 5] [0 9]] (REQUEST)
r 5.678305682 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 3 [1 5] [0 9]] (REQUEST)
r 5.678306029 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 3 [1 5] [0 9]] (REQUEST)
s 5.678306029 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 2] [0x4 1 [1 6] 10.000000] (REPLY)
r 5.678306076 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 3 [1 5] [0 9]] (REQUEST)
s 5.679947136 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [3:255 -1:255 28 0] [0x2 3 3 [1 5] [0 9]] (REQUEST)
r 5.682433791 _0_ RTR — 0 AODV 44 [13a 0 3 800] ——- [1:255 0:255 29 0] [0x4 2 [1 2] 10.000000] (REPLY)
s 5.682433791 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [0] 0 0
r 5.694526421 _2_ RTR — 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 30 2] [0x4 1 [1 12] 10.000000] (REPLY)
f 5.694526421 _2_ RTR — 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 29 3] [0x4 2 [1 12] 10.000000] (REPLY)
r 5.696474821 _3_ RTR — 0 AODV 44 [13a 3 2 800] ——- [1:255 0:255 29 3] [0x4 2 [1 12] 10.000000] (REPLY)
f 5.696474821 _3_ RTR — 0 AODV 44 [13a 3 2 800] ——- [1:255 0:255 28 2] [0x4 3 [1 12] 10.000000] (REPLY)
D 5.696474821 _3_ IFQ ARP 0 AODV 44 [13a 3 3 800] ——- [1:255 0:255 28 2] [0x4 3 [1 6] 10.000000] (REPLY)
...
r 5.700809487 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 26 0] [0x2 5 5 [1 10] [0 14]] (REQUEST)
s 5.700809487 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [2:255 0:255 30 3] [0x4 2 [1 12] 9.000000] (REPLY)
r 5.700809778 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 26 0] [0x2 5 5 [1 10] [0 14]] (REQUEST)
r 5.700809920 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 26 0] [0x2 5 5 [1 10] [0 14]] (REQUEST)
r 5.702343754 _2_ RTR — 0 AODV 44 [13a 2 3 800] ——- [1:255 0:255 28 2] [0x4 3 [1 12] 10.000000] (REPLY)
r 5.705962421 _3_ RTR — 0 AODV 44 [13a 3 2 800] ——- [2:255 0:255 30 3] [0x4 2 [1 12] 9.000000] (REPLY)
s 5.793703616 _0_ AGT — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
r 5.793703616 _0_ RTR — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
s 5.793703616 _0_ RTR — 1 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [1] 0 0
r 5.795440889 _3_ RTR — 1 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [1] 1 0
f 5.795440889 _3_ RTR — 1 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 2] [1] 1 0
r 5.797541289 _2_ RTR — 1 cbr 68 [13a 2 3 800] ——- [0:0 1:0 29 2] [1] 2 0
f 5.797541289 _2_ RTR — 1 cbr 68 [13a 2 3 800] ——- [0:0 1:0 28 1] [1] 2 0
r 5.799762731 _1_ AGT — 1 cbr 68 [13a 1 2 800] ——- [0:0 1:0 28 1] [1] 3 0
...
```

`RREQ_MF_NI`: In general, an inside attacker cannot completely isolate a node by modifying one RREQ message, but it can prevent the source node from receiving data packets from other nodes for a short period of time. After receiving a RREQ message from the source node, the attacker applies the following modifications:

1. Increase the RREQ ID by a small number;
2. Replace the destination IP address with a non-existent IP address;
3. Increase the source sequence number by at least one;
4. Set the source IP address in IP header to a non-existent IP address.

When the neighbors of the attacker receive the faked RREQ message, they update the next hop to the source node to a non-existent node, which is indicated by the source IP address in IP header of the RREQ message, since the faked RREQ message has a greater source sequence number.

Because the faked RREQ message has a non-existent destination IP address, it can be broadcasted to the farthest nodes in the ad-hoc network, and no RREP message is generated. When other nodes want to send data packets to the source node, they just use the routes established by the faked RREQ message. The data packets are dropped due to the non-existent node in the routes. The attacker may isolate a victim node from receiving from other nodes for a short period of time until new routes are estalished. Alternatively, the attacker can set its own IP address as the source IP address in the IP header, so it can receive and drop the data packets from other nodes to the victim node.

In the simulation, node 0 is the victim node. In normal situations, node 1 can send data packets to node 0 without sending a RREQ message, because it can use the reverse route added by the RREQ message originated from node 0. With the atomic misuse `RREQs_MF_NI`, node 1 fails when sending data packets to a non-existent node, and then generates a RERR message. At the same time, all nodes receiving the faked RREQ message cannot send data packets to node 0. The related fragment of the trace file is as follows:

...
s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669518888 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519456 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519512 _1_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.669519512 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
f 5.669660935 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [88:255 -1:255 29 0] [0x2 2 2 [1 0] [0 6]] (REQUEST)
s 5.670400683 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.673439043 _0_ RTR — 0 AODV 44 [13a 0 1 800] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
s 5.673439043 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 1] [0] 0 0
r 5.674763944 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.674764345 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.674764512 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.676492882 _1_ AGT — 0 cbr 68 [13a 1 0 800] ——- [0:0 1:0 30 1] [0] 1 0
r 5.677757675 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 29 0] [0x2 2 2 [1 0] [0 6]] (REQUEST)
r 5.677758076 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 29 0] [0x2 2 2 [1 0] [0 6]] (REQUEST)
r 5.677758135 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 29 0] [0x2 2 2 [1 0] [0 6]] (REQUEST)
s 5.677758135 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 88] [0x4 1 [1 2] 10.000000] (REPLY)
...
s 6.068618722 _1_ RTR — 3 cbr 68 [0 0 0 0] ——- [1:1 0:1 30 88] [0] 0 0
D 6.068618722 _1_ IFQ ARP 0 AODV 44 [0 0 1 800] ——- [1:255 0:255 30 88] [0x4 1 [1 2] 10.000000] (REPLY)
D 6.068618722 _1_ RTR CBK 3 cbr 68 [0 0 1 800] ——- [1:1 0:1 30 88] [0] 0 0
s 6.068618722 _1_ RTR — 0 AODV 32 [0 0 0 0] ——- [1:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 6.069390888 _3_ RTR — 0 AODV 32 [0 ffffffff 1 800] ——- [1:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 6.069391348 _2_ RTR — 0 AODV 32 [0 ffffffff 1 800] ——- [1:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 6.069391512 _0_ RTR — 0 AODV 32 [0 ffffffff 1 800] ——- [1:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
...

`RREQ_MF_RC`: It is difficult for an attacker to consume too much resource with one faked RREQ message. However, an attacker may still be able to introduce unnecessary broadcast messages into the network through a single `RREQ_MF_RC` misuse. Specifically, an attacker can modify an incoming RREQ message to make it appear to be fresh (by increasing the RREQ ID) so that it will be rebroadcasted by the attacker's neighbors. To generate real impact on the network, the attacker needs to repeatedly apply `RREQ_MF_RC` misuses, and generate a broadcast message loop in the network. We will discuss such misuses in the context of compound misuses.

A.1.3 Atomic misuses `RREQ_AF`

`RREQ_AF_RD`: If there exists a route from a source node to a destination node, an inside attacker can break down the route by broadcasting a faked RREQ. In the faked RREQ message, the attacker pretends to rebroadcast a RREQ message initiated from the destination node to the source node with a non-existent node as the source IP address in the IP header, just as described in `RREQ_MF_RD`. Due to the same reason described in `RREQ_MF_RD`, the source node will update its route to the destination node through a non-existent node, so the route is broken down.

In the simulation, node 0 is the victim node, and it already has a route to node 1. After node 2 sends a faked RREQ message actively, the route from node 0 to node 1 is disrupted. The fragment of the trace file is as follows:

...
s 6.133635540 _0_ AGT — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
r 6.133635540 _0_ RTR — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
s 6.133635540 _0_ RTR — 2 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [2] 0 0
r 6.135372813 _3_ RTR — 2 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [2] 1 0
f 6.135372813 _3_ RTR — 2 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 1] [2] 1 0
r 6.137574510 _1_ AGT — 2 cbr 68 [13a 1 3 800] ——- [0:0 1:0 29 1] [2] 2 0
s 6.460392679 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
r 6.460392679 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
s 6.460392679 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [3] 0 0
r 6.462129952 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [3] 1 0
f 6.462129952 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 1] [3] 1 0
r 6.464571649 _1_ AGT — 3 cbr 68 [13a 1 3 800] ——- [0:0 1:0 29 1] [3] 2 0
s 6.668618722 _2_ AGT — 4 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0
r 6.668618722 _2_ RTR — 4 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0
s 6.668618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [88:255 -1:255 0 0] [0x2 1 11 [0 4] [1 6]] (REQUEST)
r 6.669518855 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 0 0] [0x2 1 11 [0 4] [1 6]] (REQUEST)
s 6.669518855 _3_ RTR — 0 AODV 44 [0 0 0 0] ——- [3:255 1:255 30 88] [0x4 2 [0 4] 9.000000] (REPLY)
r 6.669519202 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 0 0] [0x2 1 11 [0 4] [1 6]] (REQUEST)
r 6.669519249 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 0 0] [0x2 1 11 [0 4] [1 6]] (REQUEST)
s 6.669519249 _0_ RTR — 0 AODV 44 [0 0 0 0] ——- [0:255 1:255 30 88] [0x4 1 [0 6] 10.000000] (REPLY)
s 6.715054878 _0_ AGT — 5 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
r 6.715054878 _0_ RTR — 5 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
s 6.715054878 _0_ RTR — 5 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [4] 0 0
D 6.715054878 _0_ IFQ ARP 0 AODV 44 [0 0 0 800] ——- [0:255 1:255 30 88] [0x4 1 [0 6] 10.000000] (REPLY)
...
s 7.009392113 _0_ AGT — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [5] 0 0
r 7.009392113 _0_ RTR — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [5] 0 0
s 7.009392113 _0_ RTR — 7 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [5] 0 0
D 7.009392113 _0_ IFQ ARP 5 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [4] 0 0
...

RREQ_AF_RI: An attacker can invade a route if the attacker is in the transmission range of the source node of the route. The attacker pretends to forwards a RREQ message initiated from the source node to the destination node as described in RREQ_MF_RI. Due to the same reason described in RREQ_MF_RI, after the source node receives the the RREP message forwarded by the attacker, it updates the attacker as the next hop to the destination node. The attacker needs to have a route to forward the RREP message to the source node. As a result, the attacker succeeds in invading the route from the source node to the destination node.

In the simulation, malicious node 2 succeeds in invading the route from node 0 to node 1 after applying RREQ_AF_RI. The fragment of the trace file is as follows:

...
s 0.200000000 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 0.200000000 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 0.200000000 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 0.200900547 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 0.200900567 _1_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 0.200900567 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
s 0.201042593 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 0.204453551 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 0.204453739 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 0.205969251 _0_ RTR — 0 AODV 44 [13a 0 1 800] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
...
s 0.693391010 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
r 0.693391010 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
s 0.693391010 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 1] [3] 0 0
r 0.695128710 _1_ AGT — 3 cbr 68 [13a 1 0 800] ——- [0:0 1:0 30 1] [3] 1 0
...
s 30.668618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [2:255 -1:255 30 0] [0x2 1 1 [1 12] [0 12]] (REQUEST)
r 30.669518976 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 1 [1 12] [0 12]] (REQUEST)
r 30.669519070 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 1 [1 12] [0 12]] (REQUEST)
r 30.669519243 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 1 [1 12] [0 12]] (REQUEST)
s 30.669519243 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 2] [0x4 1 [1 14] 10.000000] (REPLY)
s 30.672031970 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 12] [0 12]] (REQUEST)
r 30.673676896 _2_ RTR — 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 30 2] [0x4 1 [1 14] 10.000000] (REPLY)
D 30.673676896 _2_ RTR NRTE 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 29 2] [0x4 1 [1 14] 10.000000] (REPLY)
s 30.673676896 _2_ RTR — 128 cbr 68 [0 0 0 0] ——- [2:1 1:1 30 1] [0] 0 0
r 30.674981592 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 12] [0 12]] (REQUEST)
r 30.674981603 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 12] [0 12]] (REQUEST)
r 30.674981790 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 12] [0 12]] (REQUEST)
r 30.676749157 _1_ AGT — 128 cbr 68 [13a 1 2 800] ——- [2:1 1:1 30 1] [0] 1 0
s 30.768618722 _2_ AGT — 129 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0
r 30.768618722 _2_ RTR — 129 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0
s 30.768618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [2:255 -1:255 30 0] [0x2 1 2 [0 10] [1 14]] (REQUEST)
r 30.769518976 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 2 [0 10] [1 14]] (REQUEST)
s 30.769518976 _0_ RTR — 0 AODV 44 [0 0 0 0] ——- [0:255 1:255 30 2] [0x4 1 [0 12] 10.000000] (REPLY)
r 30.769519070 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 2 [0 10] [1 14]] (REQUEST)
s 30.769519070 _3_ RTR — 0 AODV 44 [0 0 0 0] ——- [3:255 1:255 30 2] [0x4 2 [0 12] 5.000000] (REPLY)

r 30.769519243 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 2 [0 10] [1 14]] (REQUEST)
...
s 31.198477777 _0_ AGT — 132 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [129] 0 0
r 31.198477777 _0_ RTR — 132 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [129] 0 0
s 31.198477777 _0_ RTR — 132 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 2] [129] 0 0
r 31.200214539 _2_ RTR — 132 cbr 68 [13a 2 0 800] ——- [0:0 1:0 30 2] [129] 1 0
f 31.200214539 _2_ RTR — 132 cbr 68 [13a 2 0 800] ——- [0:0 1:0 29 1] [129] 1 0
r 31.202716104 _1_ AGT — 132 cbr 68 [13a 1 2 800] ——- [0:0 1:0 29 1] [129] 2 0
...

RREQ_AF_NI: By broadcasting one faked RREQ message, an inside attacker can prevent any victim node from receiving data packets from other nodes for a short period of time, just as described in RREQ_MF_NI. But the attacker cannot completely isolate a victim node from sending data packets to other nodes unless other misuses are also used. In the simulation, node 3 cannot send data packets to node 0 using the route established by the faked RREQ message and generates a RERR message. The fragment of trace file is as follows:
s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669518888 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519065 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519139 _1_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.669519139 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
s 5.669660935 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [2:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.670400292 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.673436063 _0_ RTR — 0 AODV 44 [13a 0 1 800] ——- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)
...
s 5.793703616 _0_ AGT — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
r 5.793703616 _0_ RTR — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
s 5.793703616 _0_ RTR — 1 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 1] [1] 0 0
r 5.795440869 _1_ AGT — 1 cbr 68 [13a 1 0 800] ——- [0:0 1:0 30 1] [1] 1 0
...
s 6.268618722 _3_ AGT — 4 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [0] 0 0
r 6.268618722 _3_ RTR — 4 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [0] 0 0
s 6.268618722 _3_ RTR — 4 cbr 68 [0 0 0 0] ——- [3:0 0:2 30 0] [0] 0 0
r 6.273052124 _0_ AGT — 4 cbr 68 [13a 0 3 800] ——- [3:0 0:2 30 0] [0] 1 0
...
s 6.668618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [88:255 -1:255 30 0] [0x2 1 101 [66 4] [0 16]] (REQUEST)
r 6.669518888 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 30 0] [0x2 1 101 [66 4] [0 16]] (REQUEST)
r 6.669518924 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 30 0] [0x2 1 101 [66 4] [0 16]] (REQUEST)
r 6.669518976 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [88:255 -1:255 30 0] [0x2 1 101 [66 4] [0 16]] (REQUEST)
s 6.671931181 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [3:255 -1:255 29 0] [0x2 2 101 [66 4] [0 16]] (REQUEST)
...
s 7.096760041 _3_ AGT — 12 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [4] 0 0
r 7.096760041 _3_ RTR — 12 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [4] 0 0
s 7.096760041 _3_ RTR — 12 cbr 68 [0 0 0 0] ——- [3:0 0:2 30 88] [4] 0 0
D 7.096760041 _3_ IFQ ARP 10 cbr 68 [0 0 3 800] ——- [3:0 0:2 30 88] [3] 0 0

...
s 7.377469954 _3_ AGT — 15 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [6] 0 0
r 7.377469954 _3_ RTR — 15 cbr 48 [0 0 0 0] ——- [3:0 0:2 32 0] [6] 0 0
s 7.377469954 _3_ RTR — 15 cbr 68 [0 0 0 0] ——- [3:0 0:2 30 88] [6] 0 0
D 7.377469954 _3_ RTR CBK 13 cbr 68 [0 0 3 800] ——- [3:0 0:2 30 88] [5] 0 0
D 7.377469954 _3_ RTR CBK 15 cbr 68 [0 0 3 800] ——- [3:0 0:2 30 88] [6] 0 0
s 7.377469954 _3_ RTR — 0 AODV 32 [0 0 0 0] ——- [3:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 7.378242157 _2_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 7.378242178 _1_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
r 7.378242298 _0_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
...

`RREQ_AF_RC`: An inside attacker can only introduce limited broadcast messages into the network by a single `RREQ_AF_RC`.

A.2 Atomic Misuses of RREP Messages

A.2.1 Atomic Misuses `RREP_DR`

`RREP_DR_RD`: If during a route discovery process, only one node generates a RREP message and an attacker drops the RREP message when the only RREP message passes through the attacker, so the source node cannot receive the RREP message and the route cannot be established. The source node has to initiate another round of route discovery process. However, when the source node has multiple neighbors and not all of them are malicious, this misuse has very limited impact.

`RREP_DR_RI`: Obviously, an attacker cannot invade a route by dropping a RREP message, which is used to establish a route.

`RREP_DR_NI`: Because a node may communicate with several nodes in an ad-hoc network, and the RREP messages from different destination nodes may reach the source node through different neighbors, an inside attacker cannot isolate a node by dropping only one RREP message. However, if an attacker is the only neighbor of a victim node, it can partially isolate the victim node by dropping all the RREP messages sent to or from the victim node. This is essentially a compound misuse due to the multiple dropping actions.

`RREP_DR_RC`: `RREP_DR` misuses cannot consume noticeable resource of other nodes.

A.2.2 Atomic Misuses `RREP_MF`

`RREP_MF_RD`: In a route discovery process, if the only RREP message passes through an inside attacker, the attacker can prevent the route from being established by applying one of the following modifications:

- Change the value of type;
- Replace the destination IP address with another IP address;
- Replace the source IP address with another IP address;
- Decrease the TTL in IP header to 1;
- Decrease the lifetime field to 0;

- Replace the source IP address in the IP header with a non-existent IP address.

Because of the modifications of the RREP message, the source node will receive an invalid RREP message or no RREP message at all. As a result, the source node cannot establish a route to the destination node in this round of route discovery process. However, note that the victim node may receive RREP messages from other nodes. Thus, this misuse doesn't always work.

In the simulation, node 0 is the source node, and node 1 is the destination node. Before node 2 forwards the RREP message, it modifies the source IP address in IP header to a non-existent node (node 88). After node 0 receives the faked RREP message, it attempts to send data packets to the non-existent node, and generates a RERR message. The fragment of the trace file is as follows:

...
s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669518916 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519259 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.669660963 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [2:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.670400486 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.670561157 _0_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.670561343 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.670561765 _1_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.670561765 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 2] [0x4 1 [1 2] 10.000000] (REPLY)
r 5.674626385 _2_ RTR — 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 30 2] [0x4 1 [1 2] 10.000000] (REPLY)
f 5.674626385 _2_ RTR — 0 AODV 44 [13a 2 1 800] ——- [1:255 0:255 29 0] [0x4 2 [1 6] 10.000000] (REPLY)
r 5.678306922 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.678307035 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.678307080 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.680061506 _0_ RTR — 0 AODV 44 [13a 0 2 800] ——- [1:255 0:255 29 0] [0x4 2 [1 6] 10.000000] (REPLY)
s 5.680061506 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [0] 0 0
s 5.793703616 _0_ AGT — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
r 5.793703616 _0_ RTR — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
s 5.793703616 _0_ RTR — 1 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [1] 0 0
D 5.793703616 _0_ IFQ ARP 0 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [0] 0 0
s 5.947155678 _0_ AGT — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
r 5.947155678 _0_ RTR — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
s 5.947155678 _0_ RTR — 2 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [2] 0 0
D 5.947155678 _0_ IFQ ARP 1 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [1] 0 0
s 6.178069862 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
r 6.178069862 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
s 6.178069862 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [3] 0 0
D 6.178069862 _0_ RTR CBK 2 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [2] 0 0
D 6.178069862 _0_ RTR CBK 3 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [3] 0 0
s 6.178069862 _0_ RTR — 0 AODV 32 [0 0 0 0] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 6.178842057 _2_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 6.178842400 _3_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
s 6.326604576 _0_ AGT — 4 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
r 6.326604576 _0_ RTR — 4 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
s 6.326604576 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 4 0] [0x2 1 2 [1 7] [0 10]] (REQUEST)

...

RREP_MF_RI: If the RREP message is the only one responding to a RREQ message, an inside attacker doesn't have to do anything to invade the route when the RREP message passes through it, since it is already in the route. However, if there are other RREP messages reaching the source node, to guarantee that the RREP message through the attacker suppresses other RREP messages, the attacker may increase the destination sequence number of the RREP message by a small number. The source node will update its route table by the faked RREP message that has the greatest destination sequence number, and thus choose the route involving the attacker.

RREP_MF_NI: An inside attacker cannot isolate a node by manipulating only one RREP message. If the attacker is the only neighbor of a victim node, it can partially isolate the victim node by manipulating all the RREP messages sent to or from the victim node, which is essentially a compound misuse.

RREP_MF_RC: RREP_MF consumes little resource of the network and the other nodes.

A.2.3 Atomic Misuses RREP_FR

RREP_FR_RD: After receiving a RREQ messsage, an inside attacker may forge a RREP message as if it had a fresh enough route to the destination node. In order to suppress other legitimate RREP messages that the source node may receive from the other nodes, the attacker may forge a faked RREP message in the following way:

1. Set the destination IP address to the destination node's IP address;
2. Set the source IP address to the source node's IP address;
3. Set the source IP address in the IP header to a non-existent IP address;
4. Set the destination IP address in the IP header to the node from which the attacker receives the RREQ message;
5. Increase the destination sequence number by at least one, or decrease the hop count to 0.

The attacker unicasts the faked RREP message to the source node along the reverse route which is established by the RREQ message. After receiving the faked RREP message, the neighbor of the attacker will update the next hop to the destination node to the non-existent IP address in the IP header. Before the faked RREP message reaches the source node, the source node may have already received other legitimate RREP messages. Even in this case, the source node will update its next hop to the destination node as the neighbor from which it receives the faked RREP message, since the faked RREP has a greater destination sequence number or a smaller hop count than that in the source node's route table. As a result, the data packets from the source node will be lost, since they will eventually sent to a non-existent node.

In the simulation, node 0 is the source node, and node 1 is the destination node. After malicious node 2 receives the RREQ message, it responds a faked RREP message, which disrupts the route. The fragment of the trace file is as follows:

s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669518957 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)

s 5.669518957 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [88:255 0:255 30 0] [0x4 1 [1 10] 10.000000] (REPLY)
r 5.669518962 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.669661009 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.673069995 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.673070188 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.673070703 _1_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.673070703 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 0:255 30 3] [0x4 1 [1 2] 10.000000] (REPLY)
r 5.674584702 _0_ RTR — 0 AODV 44 [13a 0 2 800] ——- [88:255 0:255 30 0] [0x4 1 [1 10] 10.000000] (REPLY)
s 5.674584702 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [0] 0 0
r 5.680907982 _3_ RTR — 0 AODV 44 [13a 3 1 800] ——- [1:255 0:255 30 3] [0x4 1 [1 2] 10.000000] (REPLY)
f 5.680907982 _3_ RTR — 0 AODV 44 [13a 3 1 800] ——- [1:255 0:255 29 0] [0x4 2 [1 2] 10.000000] (REPLY)
r 5.683256703 _0_ RTR — 0 AODV 44 [13a 0 3 800] ——- [1:255 0:255 29 0] [0x4 2 [1 2] 10.000000] (REPLY)
s 5.793703616 _0_ AGT — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
r 5.793703616 _0_ RTR — 1 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
s 5.793703616 _0_ RTR — 1 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [1] 0 0
D 5.793703616 _0_ IFQ ARP 0 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [0] 0 0
s 6.133635540 _0_ AGT — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
r 6.133635540 _0_ RTR — 2 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [2] 0 0
s 6.133635540 _0_ RTR — 2 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [2] 0 0
D 6.133635540 _0_ IFQ ARP 1 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [1] 0 0
s 6.302336854 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
r 6.302336854 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
s 6.302336854 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [3] 0 0
D 6.302336854 _0_ RTR CBK 2 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [2] 0 0
D 6.302336854 _0_ RTR CBK 3 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [3] 0 0
s 6.302336854 _0_ RTR — 0 AODV 32 [0 0 0 0] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 6.303109090 _2_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 6.303109094 _3_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
...

`RREP_FR_RI`: If an inside attacker already has a route to the destination node, it can invade the route by unicasting a faked RREP message to the source node along the reverse route. The purpose of the attacker to still forge a RREP message is to surppress other RREP messages, possibly with shorter path from the destination node to the source node. In order to suppress other RREP messages, the attacker can increase the destination sequence number by a small number, or decreases the hop count to 1. After receiving all the RREP messages, the source node will update the destination sequence number in its route table to the one in the faked RREP message. It will also update the next hop to the destination node to the neighbor from which it receives the faked RREP message. As a result, the attacker can successfully be a part of the route from the source node to the destincation node.

In the simulation, malicious node 2 succeeds in invading the route from node 0 to node 1. The fragment of trace file is as follows:

...
s 27.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 27.669519022 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 27.669519170 _2_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 27.669519170 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [2:255 0:255 30 0] [0x4 2 [1 2] 6.000000] (REPLY)
r 27.673861309 _0_ RTR — 0 AODV 44 [13a 0 2 800] ——- [2:255 0:255 30 0] [0x4 2 [1 2] 6.000000] (REPLY)

s 27.673861309 _0_ RTR — 6 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 2] [0] 0 0
s 27.674705510 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 27.678437155 _2_ RTR — 6 cbr 68 [13a 2 0 800] ——- [0:0 1:0 30 2] [0] 1 0
f 27.678437155 _2_ RTR — 6 cbr 68 [13a 2 0 800] ——- [0:0 1:0 29 1] [0] 1 0
r 27.680479391 _1_ AGT — 6 cbr 68 [13a 1 2 800] ——- [0:0 1:0 29 1] [0] 2 0
s 27.858140709 _0_ AGT — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
r 27.858140709 _0_ RTR — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [1] 0 0
s 27.858140709 _0_ RTR — 7 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 2] [1] 0 0
r 27.859878054 _2_ RTR — 7 cbr 68 [13a 2 0 800] ——- [0:0 1:0 30 2] [1] 1 0
f 27.859878054 _2_ RTR — 7 cbr 68 [13a 2 0 800] ——- [0:0 1:0 29 1] [1] 1 0
r 27.861960290 _1_ AGT — 7 cbr 68 [13a 1 2 800] ——- [0:0 1:0 29 1] [1] 2 0
...

`RREP_FR_NI`: A node may broadcast RREQ messages to establish routes to different destination nodes. An inside attacker cannot isolate a node by replying only one faked RREP message.

`RREP_FR_RC`: This misuse consumes little resource of the network and other nodes. Note that the impact of `RREP_FR_RC` is different from misuses by forging a RREQ message; RREQ messages are brodcasted throughout the network, while RREP messages are unicasted through a reverse route.

A.2.4 Atomic Misuses `RREP_AF`

`RREP_AF_RD`: Before the attacker launches the misuse, there exists a route from a source node to a destination node. In order to disrupt this route, the attacker can forge a RREP message as follows:

1. Set the type field to 2;
2. Set the hop count field to 1;
3. Set the source IP address as the source node of the route and the destination IP address as the destination node of the route;
4. Increase the destination sequence number by at least one;
5. Set the source IP address in the IP header to a non-existent IP address (node 88);

Suppose the attacker already has a route to the source node, it can unicast the faked RREP message to the source node. When the source node receives the faked RREP message, it will update its route to the destination node through the non-existent node for the same reason as described in `RREP_FR_DR`. In the simulation, node 0 is the source node, and node 1 is the destination node. After node 0 receives the faked RREP message initiated by node 2, the route between node 0 and node 1 is disrupted. The fragment of trace file is as follows:
...
s 6.460392679 _0_ AGT — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
r 6.460392679 _0_ RTR — 3 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [3] 0 0
s 6.460392679 _0_ RTR — 3 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [3] 0 0
r 6.462129952 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [3] 1 0
f 6.462129952 _3_ RTR — 3 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 1] [3] 1 0
r 6.464571649 _1_ AGT — 3 cbr 68 [13a 1 3 800] ——- [0:0 1:0 29 1] [3] 2 0
s 6.668618722 _2_ AGT — 4 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0
r 6.668618722 _2_ RTR — 4 cbr 48 [0 0 0 0] ——- [2:0 0:1 32 0] [0] 0 0

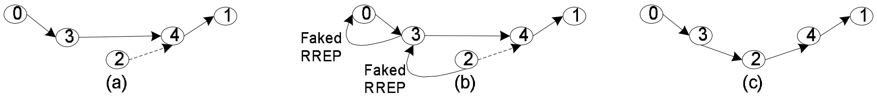


Figure 4: An Attacker Invades A Route by Sending A Faked RREP Actively.

s 6.668618722 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [88:255 0:255 30 0] [0x4 1 [1 10] 10.000000] (REPLY)
r 6.673061411 _0_ RTR — 0 AODV 44 [13a 0 2 800] ——- [88:255 0:255 30 0] [0x4 1 [1 10] 10.000000] (REPLY)
s 6.715054878 _0_ AGT — 5 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
r 6.715054878 _0_ RTR — 5 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [4] 0 0
s 6.715054878 _0_ RTR — 5 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [4] 0 0
s 7.000000000 _2_ RTR — 4 cbr 68 [0 0 0 0] ——- [2:0 0:1 30 0] [0] 0 0
r 7.001737581 _0_ AGT — 4 cbr 68 [13a 0 2 800] ——- [2:0 0:1 30 0] [0] 1 0
s 7.009392113 _0_ AGT — 6 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [5] 0 0
r 7.009392113 _0_ RTR — 6 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [5] 0 0
s 7.009392113 _0_ RTR — 6 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [5] 0 0
D 7.009392113 _0_ IFQ ARP 5 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [4] 0 0
s 7.235771883 _0_ AGT — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [6] 0 0
r 7.235771883 _0_ RTR — 7 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [6] 0 0
s 7.235771883 _0_ RTR — 7 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [6] 0 0
D 7.235771883 _0_ IFQ ARP 6 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [5] 0 0
s 7.461823765 _0_ AGT — 8 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [7] 0 0
r 7.461823765 _0_ RTR — 8 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [7] 0 0
s 7.461823765 _0_ RTR — 8 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 88] [7] 0 0
D 7.461823765 _0_ RTR CBK 7 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [6] 0 0
D 7.461823765 _0_ RTR CBK 8 cbr 68 [0 0 0 800] ——- [0:0 1:0 30 88] [7] 0 0
s 7.461823765 _0_ RTR — 0 AODV 32 [0 0 0 0] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 7.462596189 _3_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 7.462596292 _2_ RTR — 0 AODV 32 [0 ffffffff 0 800] ——- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
s 7.808684651 _0_ AGT — 9 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [8] 0 0
r 7.808684651 _0_ RTR — 9 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [8] 0 0
s 7.808684651 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 3 0] [0x2 1 2 [1 11] [0 10]] (REQUEST)
r 7.809585075 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:25
...

`RREP_AF_RI`: If an inside attacker has routes to both of the source node and the destination node of a route, it can invade the route by sending one faked RREP message to the source node, as Figure 4 shows.

As Figure 4 (a) shows, at the beginning, there exists a route between node 0 and node 1, and node 3 and node 4 are two intermediate nodes in the route. Node 2 is an inside attacker, which already has a route to both node 0 and node 1. In order to invade the route, the attacker pretends to forward a RREP message from node 1 to node 0. The attacker sends the faked RREP message to node 3, which forwards the faked RREP message to node 0, as Figure 4(b) shows.

The attacker can forge the RREP message in the following way:

1. Set the source IP address to node 0;

2. Set the destination IP address to node 1;

3. Set the destination sequence number to node 1's sequence number plus at least one;
4. Set the source IP address in the IP header to node 2;
5. Set the destination IP address in the IP header to node 3.

When node 3 and node 0 receive the faked RREP message, they will update the sequence number of node 1 in their route tables to the destination sequence number in the faked RREP message. Node 0 still uses node 3 as the next hop to node 1, but node 3 updates node 2 as the next hop to node 1. Because node 2 already has a route to node 1, it can forward the data packets from node 0 to node 1, as Figure 4(c) shows.

In the simulation, before the misuse, the route is $0 \Rightarrow 3 \Rightarrow 4 \Rightarrow 1$; after malicious node 2 actively sends out a faked RREP message,the route changes to $0 \Rightarrow 3 \Rightarrow 2 \Rightarrow 4 \Rightarrow 1$. The fragment of the trace file is as follows:

```
...
s 7.693402932 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [0] 0 0
r 7.695563792 _3_ RTR — 0 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [0] 1 0
f 7.695563792 _3_ RTR — 0 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 4] [0] 1 0
r 7.697885792 _4_ RTR — 0 cbr 68 [13a 4 3 800] ——- [0:0 1:0 29 4] [0] 2 0
f 7.697885792 _4_ RTR — 0 cbr 68 [13a 4 3 800] ——- [0:0 1:0 28 1] [0] 2 0
r 7.700206910 _1_ AGT — 0 cbr 68 [13a 1 4 800] ——- [0:0 1:0 28 1] [0] 3 0
...
s 10.671253994 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [2:255 0:255 30 3] [0x4 1 [1 7] 10.000000] (REPLY)
r 10.680841103 _3_ RTR — 0 AODV 44 [13a 3 2 800] ——- [2:255 0:255 30 3] [0x4 1 [1 7] 10.000000] (REPLY)
f 10.680841103 _3_ RTR — 0 AODV 44 [13a 3 2 800] ——- [2:255 0:255 29 0] [0x4 2 [1 7] 10.000000] (REPLY)
r 10.682829964 _0_ RTR — 0 AODV 44 [13a 0 3 800] ——- [2:255 0:255 29 0] [0x4 2 [1 7] 10.000000] (REPLY)
...
s 11.619037018 _0_ AGT — 17 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [15] 0 0
r 11.619037018 _0_ RTR — 17 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [15] 0 0
s 11.619037018 _0_ RTR — 17 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [15] 0 0
r 11.620773878 _3_ RTR — 17 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [15] 1 0
f 11.620773878 _3_ RTR — 17 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 2] [15] 1 0
r 11.623295681 _2_ RTR — 17 cbr 68 [13a 2 3 800] ——- [0:0 1:0 29 2] [15] 2 0
f 11.623295681 _2_ RTR — 17 cbr 68 [13a 2 3 800] ——- [0:0 1:0 28 4] [15] 2 0
r 11.625537484 _4_ RTR — 17 cbr 68 [13a 4 2 800] ——- [0:0 1:0 28 4] [15] 3 0
f 11.625537484 _4_ RTR — 17 cbr 68 [13a 4 2 800] ——- [0:0 1:0 27 1] [15] 3 0
r 11.627638602 _1_ AGT — 17 cbr 68 [13a 1 4 800] ——- [0:0 1:0 27 1] [15] 4 0
...
```

`RREP_AF_NI`: An inside attacker cannot isolate a victim node by sending out only one faked RREP message.

`RREP_AF_RC`: An inside attacker can form a loop in the network to consume resources of the nodes in the loop. As Figure 5 shows, there are two intermediate nodes, node 3 and node 4, in a route from node 0 to node 1.

The attacker can form a data packets loop between node 3 and node 4 by pretending to be node 3 to forward a RREP message from the destination node 1 to the source node 0. The faked RREP message is generated as follows:

1. Set the destination IP address to node 1;

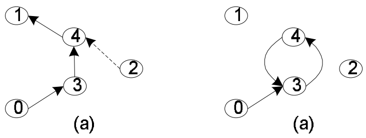


Figure 5: The attacker forms a loop between node 3 and node 4 by a faked RREP message. 0: source node; 1:destination node; 2: Attacker; 3,4: intermediate nodes.

2. Set the destination sequence number as node 1's sequence number plus at least one;
3. Set the source IP address to node 0;
4. Set the source IP address in the IP header to node 3;
5. Set the destination IP address in the IP header to node 4.

When node 4 receives the faked RREP message, it updates the next hop to node 1 as node 3. Since there is still an entry in node 4's route table that indicates the next hop to node 0 is node 3, node 4 will forward the faked RREP message to node 3, which will then forward the faked RREP message to node 0. After updating the destination sequence number in the route table, if node 0 continues to send data packets to node 1, these packets will be first sent to node 3, then node 4, and finally back to node 3 again. As a result, a loop is formed between node 3 and node 4. These data packets will be dropped until the TTL fields in the IP packets decrease to 0. The fragment of the trace file is as follows:

```
...
s 10.463296343 _0_ AGT — 11 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [11] 0 0
r 10.463296343 _0_ RTR — 11 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [11] 0 0
s 10.463296343 _0_ RTR — 11 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [11] 0 0
r 10.465033204 _3_ RTR — 11 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [11] 1 0
f 10.465033204 _3_ RTR — 11 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 4] [11] 1 0
r 10.467575204 _4_ RTR — 11 cbr 68 [13a 4 3 800] ——- [0:0 1:0 29 4] [11] 2 0
f 10.467575204 _4_ RTR — 11 cbr 68 [13a 4 3 800] ——- [0:0 1:0 28 1] [11] 2 0
r 10.469876322 _1_ AGT — 11 cbr 68 [13a 1 4 800] ——- [0:0 1:0 28 1] [11] 3 0
s 10.668618722 _2_ AGT — 12 cbr 48 [0 0 0 0] ——- [2:0 4:0 32 0] [0] 0 0
r 10.668618722 _2_ RTR — 12 cbr 48 [0 0 0 0] ——- [2:0 4:0 32 0] [0] 0 0
s 10.668618722 _2_ RTR — 0 AODV 44 [0 0 0 0] ——- [3:255 0:255 30 4] [0x4 1 [1 10] 10.000000] (REPLY)
r 10.670164525 _4_ RTR — 0 AODV 44 [13a 4 2 800] ——- [3:255 0:255 30 4] [0x4 1 [1 10] 10.000000] (REPLY)
f 10.670164525 _4_ RTR — 0 AODV 44 [13a 4 2 800] ——- [3:255 0:255 29 3] [0x4 2 [1 10] 10.000000] (REPLY)
r 10.672554525 _3_ RTR — 0 AODV 44 [13a 3 4 800] ——- [3:255 0:255 29 3] [0x4 2 [1 10] 10.000000] (REPLY)
s 10.672554525 _3_ RTR — 0 AODV 44 [13a 3 4 800] ——- [3:255 0:255 28 0] [0x4 3 [1 10] 10.000000] (REPLY)
r 10.674543385 _0_ RTR — 0 AODV 44 [13a 0 3 800] ——- [3:255 0:255 28 0] [0x4 3 [1 10] 10.000000] (REPLY)
s 10.792191929 _0_ AGT — 13 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [12] 0 0
r 10.792191929 _0_ RTR — 13 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [12] 0 0
s 10.792191929 _0_ RTR — 13 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [12] 0 0
r 10.793928789 _3_ RTR — 13 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [12] 1 0
f 10.793928789 _3_ RTR — 13 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 4] [12] 1 0
r 10.796050789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 29 4] [12] 2 0
f 10.796050789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 28 3] [12] 2 0
```

r 10.798272789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 28 3] [12] 3 0
f 10.798272789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 27 4] [12] 3 0
r 10.800414789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 27 4] [12] 4 0
f 10.800414789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 26 3] [12] 4 0
r 10.802636789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 26 3] [12] 5 0
f 10.802636789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 25 4] [12] 5 0
r 10.804738789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 25 4] [12] 6 0
f 10.804738789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 24 3] [12] 6 0
...
f 10.852782789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 2 3] [12] 28 0
r 10.854884789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 2 3] [12] 29 0
f 10.854884789 _3_ RTR — 13 cbr 68 [13a 3 4 800] ——- [0:0 1:0 1 4] [12] 29 0
r 10.856986789 _4_ RTR — 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 1 4] [12] 30 0
D 10.856986789 _4_ RTR TTL 13 cbr 68 [13a 4 3 800] ——- [0:0 1:0 0 4] [12] 30 0
...

A.3 Atomic Misuses of RERR Messages

A.3.1 Atomic Misuses `RERR_DR`

Atomic misuses `RERR_DR` refer to the misuses with which an attacker simply drops a RERR message it receives without notifying its neighbors in the precursor list.

`RERR_DR_RD`: Suppose an inside attacker is the only neighbor in the precursor list of a node that sends a RERR message. If the attacker drops the RERR message, the upstream nodes in the precusor list of the attacker cannot receive the RERR message, and they won't be able to notify their upstream nodes about the broken link. These upstream nodes continue to send data packets through the broken route, and these data packets are dropped due to the broken link. However, because the node that drops the data packets may send out other RERR messages and this atomic misuse only drops one RERR message, this goal can only last for a short time.

`RERR_DR_RI`: An attacker cannot invade a route by dropping a RERR message.

`RERR_DR_NI`: An attacker cannot isolate a victim node by dropping a RERR message.

`RERR_DR_RC`: If the upstream nodes of the attacker cannot receive the RERR message, they continue to use the broken route to send data packets. However, it cannot consume too much resource of the network and the other nodes.

A.3.2 Atomic Misuses `RERR_MF`

`RERR_MF_RD`: By receiving and modifying a RERR message, an inside attacker can disrupt several routes that involve the attacker. In the faked RERR message, the attacker may replace an unreachable destination IP address with another IP address, or append new unreachable Destination IP addresses that, in fact, can be reached through the attacker. The attacker needs to increment the unreachable destination sequence number by at least one, and then broadcasts the faked RERR message to all its neighbors. If a neighbor has a route to an unreachable destination node in the faked RERR message and the next hop equals to the attacker (indicated by the source IP address in the IP header), it disables this route and updates the destination sequence number with the unreachable destination sequence number in the faked RERR message. The neighbors will then forward the faked RERR message to its neighbors in their percursor lists. As a result, all the nodes that have a route through the attacker to the destination node will disable the route. Atomic misuses `RERR_MF` need to be triggered by the receipt of a RERR message before modifying and forwarding it to other nodes. In fact, an attacker can send out a faked RERR message without receiving any RERR message, as we will discuss in atomic misuses `RERR_AF`.

`RERR_MF_RI`: An insider attacker cannot invade a route using this atomic misuses.

`RERR_MF_NI`: If an inside attacker is the only neighbor of a victim node, it can disable all the route entries in the victim node's routing table by sending one faked RERR messsage. When the attacker receives a RERR message, it appends all the destination nodes in its route table into the RERR message and increase the corresponding unreachable destination sequence numbers by at least one. The attacker unicasts the faked RERR message to the victim node. When the victim node receives the faked RERR message, since all of its route entries use the attacker as the next hop and the unreachable destination sequence numbers in the faked RERR message are greater than the corresponding destination nodes' sequence numbers in its route table, it disables all the route entries in its route table. In `RERR_MF_NI`, the attacker needs to receive a RERR message before sending out a faked RERR message, so its effect is quite limited.

`RERR_MF_RC`: From `RERR_MF_RD`, we know that one faked RERR message may effect several nodes in such way that each node may invalidate serveral route entries in its route table. When they want to send data packets to those destination nodes but have no valid routes, they have to send RREQ messages to establish the routes again. One faked RERR message may cause several RREQ messages broadcasted in the whole ad-hoc network, so we consider that this misuse succeeds in consuming the resource of the network and the other nodes.

A.3.3 Atomic Misuses `RERR_AF`

`RERR_AF_RD`: It's easy to see that an inside attacker may disrupt a route by sending out one faked RERR message. If the attacker is in the transmission range of an intermediate node of a route, the attacker may impersonate the intermediate node to broadcast a faked RERR message. The attacker may forge such a RERR message in the following way:

1. Set the route's destination node as the unreachable destination address;
2. Set the intermediate node's IP address as the source IP address in IP header;
3. Set the unreachable destination sequence number as a number greater than the destination node's sequence number.

The attacker broadcasts the faked RERR message to its neighbors. If a neigbhor has a route to the destination node with the impersonated node as the next hop, it will disable the corresponding route entry. In addition, it will forward the RERR message to its upstream neighbors in its precursor list. As a result, the routes through the intermediate node to the destination node will be disrupted.

In the simulation, malicious node 2 sends a RERR message to disable the route from node 0 to node 1. The fragment of the trace file is as follows:

...
s 30.556909022 _0_ AGT — 119 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [119] 0 0
r 30.556909022 _0_ RTR — 119 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [119] 0 0
s 30.556909022 _0_ RTR — 119 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [119] 0 0

r 30.558645882 _3_ RTR — 119 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [119] 1 0
f 30.558645882 _3_ RTR — 119 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 4] [119] 1 0
r 30.560767882 _4_ RTR — 119 cbr 68 [13a 4 3 800] ——- [0:0 1:0 29 4] [119] 2 0
f 30.560767882 _4_ RTR — 119 cbr 68 [13a 4 3 800] ——- [0:0 1:0 28 1] [119] 2 0
r 30.562849000 _1_ AGT — 119 cbr 68 [13a 1 4 800] ——- [0:0 1:0 28 1] [119] 3 0
...
f 30.768618722 _2_ RTR — 0 AODV 32 [0 0 0 0] ——- [4:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 30.769391323 _4_ RTR — 0 AODV 32 [0 ffffffff 2 800] ——- [4:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 30.769391323 _3_ RTR — 0 AODV 32 [0 ffffffff 2 800] ——- [4:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
s 30.774151077 _3_ RTR — 0 AODV 32 [0 0 0 0] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 30.774923364 _0_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 30.774923678 _2_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
r 30.774923744 _4_ RTR — 0 AODV 32 [0 ffffffff 3 800] ——- [3:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)
...
s 30.790606169 _0_ AGT — 121 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [120] 0 0
r 30.790606169 _0_ RTR — 121 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [120] 0 0
s 30.790606169 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 5 0] [0x2 1 2 [1 4] [0 6]] (REQUEST)
...

RERR_AF_RI: In inside attacker cannot invade a route by sending a faked RERR message.

RERR_AF_NI: An inside attacker can launch this atomic misuse in the same way as described in RERR_MF_NI except that the attacker needs not to receive any RERR message in advance. The faked RERR message can only prevent the victim node from sending data packets for a short period of time, and it cannot prevent the victim node from receiving data packets. Moreover, to be successful, RERR_AF_NI requires the attacker to be the only neighbor of the victim node.

RERR_AF_RC: This atomic misuse can consume the resource of the network and other nodes to a certain degree. A victim node disables several routes after receiving a faked RERR message, as described in RERR_AF_RD. If the victim node still needs to send data packets to the corresponding destination nodes, it has to broadcast a RREQ message to establish a new route. One faked RERR message may make several nodes broadcast several RREQ messages to establish new valid routes to the destination node.

B Description of Compound Misuses and the Simulation Results

From the analysis of atomic misuse, we see that an inside attacker can achieve specific misuse goals; however, some of these goals may only last for a short period of time, such as RREQ_DR_RD. In order to maintain the misuses in effect, the attacker may repeat the same kind of atomic misuses many times. Besides the misuse goals that an atomic misuse may achieve, the compound misuses may achieve more misuse goals due to the change in quantity. Table 7 lists the misuse goals that compound misuses can achieve.

RREQs_DR_RD: In RREQ_DR_RD, if an inside attacker is the only node that connects two parts of the ad-hoc network, it can prevent a route from begin established between two parts of an ad-hoc network. However, this misuse goal can only last for a short period of time because the source node may send other RREQ messages to establish the route. In RREQs_DR_RD, the attacker drops all the RREQ messages sent from the source node, so the destination node cannot receive any RREQ message, and no RREP message will be generated.

Table 7: Compound Misuses of the same kind of Route Messages

Compound Misuse	Route Disruption	Route Invasion	Node Isolation	Resource Consumption
RREQs_DR	Yes (in some cases)	No	No	No
RREQs_MF	Yes	Yes	Partial	Yes
RREQs_AF	Yes	Yes	Partial	Yes
RREPs_DR	Yes (in some cases)	No	No	No
RREPs_MF	Yes	Yes	No	Yes
RREPs_FR	Yes	Yes	Partial	No
RREPs_AF	Yes	Yes	Partial	Yes
RERRs_DR	Yes (in some cases)	No	No	No
RERRs_MF	Yes	No	Partial	Yes
RERRs_AF	Yes	No	Partial	Yes

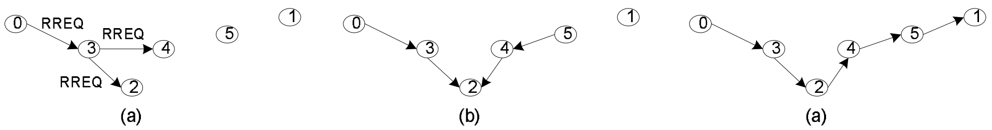


Figure 6: Route Invasion by `RREQs_MF_RI`.

`RREQs_MF_RD`: If there are alternative paths for a RREQ message to reach the destination node, the attacker can use `RREQs_MF_RD` to disrupt the route for a long period of time. When the attacker receives a RREQ message, it may launch a `RREQ_MF_RD` misuses. If the source node discovers that the route is broken, it will send out other RREQ messages to estalish the route again. Whenever the attacker receives one of such RREQ messages, it disrupts the routes by one `RREQ_MF_RD`. The simulation result of `RREQs_MF_RD` is shown in Figure 3(a). It clearly shows that the number of data packets between two nodes drops almost to zero.

`RREQs_MF_RI`: When the attacker is not in the transmission range of a source node, i.e., there exists at least one intermediate node between the attacker and the source node, the attacker still can invade the route by sending out two RREQ messages after receiving a RREQ message.

Consider the scenario shown in Figure 6(a). Suppose node 2 is a malicious node, and all the other nodes are normal. When the attacker receives a RREQ message, it may forge the first RREQ message as follows:

- Set the source IP address as node 1;
- Set the destination IP address as node 0;
- Set the source sequence number to a number greater than node 1's current sequence number;
- Set the source IP address in IP header as node 2.

Node 2 broadcasts the faked RREQ message. After receiving this message, nodes 3 and 4 will both set node 2 as the next hop to node 1, as Figure 6(b) shows. At this time, node 2 may have no route to the destination node 1. In addition, even if node 2 already has a route to node 1, there exists a loop between node 2 and node 4. In order to further establish a route from node 2 to node 1, the attacker may generate a second RREQ message as follows:

- Set the source IP address as node 2;

- Set the destination IP address as node 1;
- Set the destination sequence number to a number greater than node 1's current sequence number;
- Set the source IP address in the IP header as node 2.

Node 2 then broadcasts the second faked RREQ message. When node 4 receives this RREQ message, because it has no fresh enough route to node 1, it just rebroadcasts the RREQ message. When node 1 receives the RREQ message, it generates a RREP message which will be forwarded back to node 2. As a result, node 2 establishes a route to node 1. Now the attacker (node 2) is a part of the route from node 0 to node 1, as Figure 6(c) shows.

The simulation result of `RREQs_MF_RI` is shown in Figure 3(c). It clearly shows that misuse `RREQs_MF_RI` effectively makes the attacker a part of the route between two victim nodes.

`RREQs_MF_NI`: For the same reason as described in tt RREQ_MF_NI, an inside attacker may prevent a victim node from receiving data packets from other nodes. The faked RREQ message can suppress the legitimate RREP or RREQ messages originated from the victim node.

Local repair can prevent this misuse to some extent. In the faked RREQ messages, if the attacker sets the source IP address in the IP header as a non-existent node, the upstream node of the broken link will launch a local repair process and re-establish a route to the victim node. Because this upstream node may cache the data packets from other nodes to the victim nodes for some time[6], after establishing the route to the victim node, these data packets can be forwarded to it.

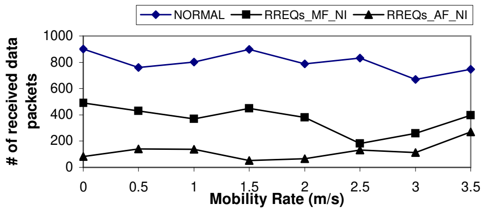
The simulation result of `RREQs_MF_NI` is shown in Figure 7(a). We can see that `RREQs_MF_NI` is not effective in isolating the victim node, the main reason is due to the random delay[7] before a node broadcasts or forwards a RREQ message. Before the faked RREQ messages reaches other nodes, other nodes may have established a route to the victim node and sent out data packets to the victim node.

`RREQs_MF_RC`: An inside attacker may form a RREQ broadcast flooding by modifying and rebroadcasting RREQ messages. Specifically, whenever an inside attacker receives a RREQ message, it increases the RREQ ID and/or source IP address to make the RREQ message appear to be fresh, and rebroadcasts the RREQ message to its neighbors. Because a node will drop a RREQ message if the TTL field in the IP header decreases to 0, the attacker also needs to reset the TTL to the maximum value. The neighbors will accept and rebroadcast the faked RREQ messages. When the attacker receives the faked RREQ messages from its neighbors, it repeats the same action as described earlier. As a result, each valid RREQ message will be forged and rebroadcasted many times. By repeating this manipulation, the attacker can form a RREQ broadcast flooding in the ad-hoc network, thus consuming the network bandwidth as well as other nodes' resources.

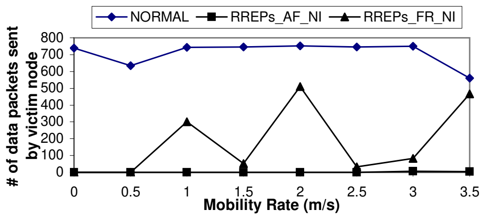
The simulation result of `RREQs_MF_RC` is shown in Figure 3(d). In the simulation, node 2 continues to increase the RREQ ID and forward the RREQ message. The fragment of the trace file is as follows:

...
s 5.668618722 _0_ AGT — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
r 5.668618722 _0_ RTR — 0 cbr 48 [0 0 0 0] ——- [0:0 1:0 32 0] [0] 0 0
s 5.668618722 _0_ RTR — 0 AODV 48 [0 0 0 0] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
r 5.669519388 _3_ RTR — 0 AODV 48 [0 ffffffff 0 800] ——- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)

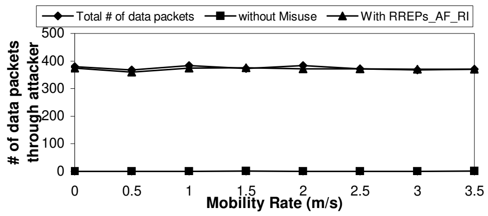
[6]In the CMU wireless extensions to ns2, the default cache size is 64, and the default cache time is 30 seconds.
[7]The average delay for broadcasting RREQ message is 10ms.



(a) Node Isolation by RREQs



(b) Node Isolation by RREPs



(c) Route Invasion by RREPs_AF_RI

Figure 7: Experimental Results about Compound Misuses

```
r 5.669519467 _2_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 1 [1 0] [0 4]] (REQUEST)
s 5.669661435 _3_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ------- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.670400694 _2_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ------- [2:255 -1:255 29 0] [0x2 2 3 [1 0] [0 4]] (REQUEST)
r 5.670561768 _2_ RTR --- 0 AODV 48 [0 ffffffff 3 800] ------- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.670562102 _1_ RTR --- 0 AODV 48 [0 ffffffff 3 800] ------- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
r 5.670562102 _0_ RTR --- 0 AODV 48 [0 ffffffff 3 800] ------- [3:255 -1:255 29 0] [0x2 2 1 [1 0] [0 4]] (REQUEST)
s 5.670562102 _1_ RTR --- 0 AODV 44 [0 0 0 0] ------- [1:255 0:255 30 3] [0x4 1 [1 2] 10.000000] (REPLY)
r 5.674625768 _3_ RTR --- 0 AODV 44 [13a 3 1 800] ------- [1:255 0:255 30 3] [0x4 1 [1 2] 10.000000] (REPLY)
f 5.674625768 _3_ RTR --- 0 AODV 44 [13a 3 1 800] ------- [1:255 0:255 29 0] [0x4 2 [1 2] 10.000000] (REPLY)
r 5.678308102 _3_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 29 0] [0x2 2 3 [1 0] [0 4]] (REQUEST)
s 5.678308102 _3_ RTR --- 0 AODV 44 [0 0 0 0] ------- [3:255 0:255 30 0] [0x4 2 [1 2] 9.000000] (REPLY)
r 5.678308435 _4_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 29 0] [0x2 2 3 [1 0] [0 4]] (REQUEST)
r 5.678308514 _1_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 29 0] [0x2 2 3 [1 0] [0 4]] (REQUEST)
r 5.678308514 _0_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 29 0] [0x2 2 3 [1 0] [0 4]] (REQUEST)
...
r 5.753825234 _2_ RTR --- 0 AODV 48 [0 ffffffff 4 800] ------- [4:255 -1:255 16 0] [0x2 15 15 [1 0] [0 4]] (REQUEST)
r 5.756005567 _0_ RTR --- 0 AODV 44 [13a 0 3 800] ------- [3:255 0:255 30 0] [0x4 2 [1 2] 9.000000] (REPLY)
r 5.758320900 _3_ RTR --- 0 AODV 44 [13a 3 1 800] ------- [1:255 0:255 30 3] [0x4 1 [1 2] 10.000000] (REPLY)
s 5.759154219 _2_ RTR --- 0 AODV 48 [0 ffffffff 4 800] ------- [2:255 -1:255 15 0] [0x2 16 17 [1 0] [0 4]] (REQUEST)
r 5.760054552 _3_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 15 0] [0x2 16 17 [1 0] [0 4]] (REQUEST)
s 5.760054552 _3_ RTR --- 0 AODV 44 [0 0 0 0] ------- [3:255 0:255 30 0] [0x4 2 [1 2] 9.000000] (REPLY)
r 5.760054886 _4_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 15 0] [0x2 16 17 [1 0] [0 4]] (REQUEST)
r 5.760054964 _1_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 15 0] [0x2 16 17 [1 0] [0 4]] (REQUEST)
r 5.760054964 _0_ RTR --- 0 AODV 48 [0 ffffffff 2 800] ------- [2:255 -1:255 15 0] [0x2 16 17 [1 0] [0 4]] (REQUEST)
...
```

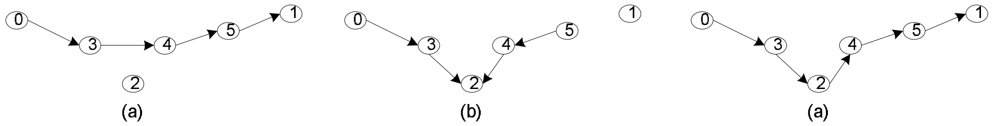


Figure 8: Route Invasion by `RREQs_AF_RI`.

`RREQs_AF_RD`: An attacker can disrupt a route using `RREQ_AF_RD`. After the source node notices that the route is broken, it may initiate another route discovery process. To make the effect persistent, the attacker needs to disrupt the routes by repeating `RREQ_AF_RD` many times. In our simulation, the attacker repeatedly sends out faked RREQ messages in a fixed interval (20pkt/s) to disable the routes between two nodes. The simulation result is shown in Figure 3(a).

`RREQs_AF_RI`: An inside attacker may invade into a route by sending out two RREQ messages actively. Consider the scenario shown in Figure 8(a). Suppose node 2 is a malicious node, and the other nodes are normal. Further assume there is a route from node 0 to node 1 through nodes 3, 4, and 5. The attacker at node 2 may forge the first RREQ message as follows:

- Set the source IP address as node 1;
- Set the destination IP address as node 0;
- Set the source sequence number to a number greater than node 1's current sequence number;
- Set the source IP address in IP header as node 2.

Node 2 may then broadcast the faked RREQ message. After receiving this message, nodes 3 and 4 will both set node 2 as the next hop to node 1, as Figure 8(b) shows. At this time, node 2 may have no route to the destination node 1. In addition, even if node 2 already has a route to node 1, there exists a loop between node 2 and node 4. In order to further establish the route from node 2 to node 1, the attacker may generate the second RREQ message as follows:

- Set the source IP address as node 2;
- Set the destination IP address as node 1;
- Set the destination sequence number to a number greater than node 1's current sequence number;
- Set the source IP address in the IP header as node 2.

Node 2 broadcasts the second faked RREQ message, when node 4 receives this RREQ message, because it has no fresh enough route to node 1, it just rebroadcasts the RREQ message. When node 1 receives the RREQ message, it generates a RREP message which will be forwarded back to node 2. So node 2 establishes a route to node 1, as Figure 8(c) shows.

The fragment of the trace file is as follows:

...

s 0.229922340 _0_ RTR — 0 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [0] 0 0

r 0.232123200 _3_ RTR — 0 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [0] 1 0

```
f 0.232123200 _3_ RTR — 0 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 4] [0] 1 0
r 0.234145200 _4_ RTR — 0 cbr 68 [13a 4 3 800] ——- [0:0 1:0 29 4] [0] 2 0
f 0.234145200 _4_ RTR — 0 cbr 68 [13a 4 3 800] ——- [0:0 1:0 28 5] [0] 2 0
r 0.236206318 _5_ RTR — 0 cbr 68 [13a 5 4 800] ——- [0:0 1:0 28 5] [0] 3 0
f 0.236206318 _5_ RTR — 0 cbr 68 [13a 5 4 800] ——- [0:0 1:0 27 1] [0] 3 0
r 0.238648318 _1_ AGT — 0 cbr 68 [13a 1 5 800] ——- [0:0 1:0 27 1] [0] 4 0
...
s 30.668618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [2:255 -1:255 30 0] [0x2 1 1 [0 0] [1 12]] (REQUEST)
r 30.669519323 _4_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 1 [0 0] [1 12]] (REQUEST)
r 30.669519323 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 1 [0 0] [1 12]] (REQUEST)
s 30.669672137 _4_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [4:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.670572510 _5_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.670572738 _2_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.670572804 _3_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
s 30.672068172 _5_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [5:255 -1:255 28 0] [0x2 3 1 [0 5] [1 12]] (REQUEST)
r 30.672968545 _4_ RTR — 0 AODV 48 [0 ffffffff 5 800] ——- [5:255 -1:255 28 0] [0x2 3 1 [0 5] [1 12]] (REQUEST)
r 30.672968839 _1_ RTR — 0 AODV 48 [0 ffffffff 5 800] ——- [5:255 -1:255 28 0] [0x2 3 1 [0 5] [1 12]] (REQUEST)
s 30.677045310 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.677945596 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
s 30.677945596 _0_ RTR — 0 AODV 44 [0 0 0 0] ——- [0:255 1:255 30 3] [0x4 1 [0 6] 10.000000] (REPLY)
r 30.677945911 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.677945976 _4_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 1 [0 5] [1 12]] (REQUEST)
r 30.679490457 _3_ RTR — 0 AODV 44 [13a 3 0 800] ——- [0:255 1:255 30 3] [0x4 1 [0 6] 10.000000] (REPLY)
f 30.679490457 _3_ RTR — 0 AODV 44 [13a 3 0 800] ——- [0:255 1:255 29 2] [0x4 2 [0 6] 10.000000] (REPLY)
r 30.684037663 _2_ RTR — 0 AODV 44 [13a 2 3 800] ——- [0:255 1:255 29 2] [0x4 2 [0 6] 10.000000] (REPLY)
D 30.684037663 _2_ RTR NRTE 0 AODV 44 [13a 2 3 800] ——- [0:255 1:255 28 2] [0x4 2 [0 6] 10.000000] (REPLY)
...
s 30.742036536 _0_ RTR — 124 cbr 68 [0 0 0 0] ——- [0:0 1:0 30 3] [123] 0 0
r 30.743773397 _3_ RTR — 124 cbr 68 [13a 3 0 800] ——- [0:0 1:0 30 3] [123] 1 0
f 30.743773397 _3_ RTR — 124 cbr 68 [13a 3 0 800] ——- [0:0 1:0 29 2] [123] 1 0
r 30.746095199 _2_ RTR — 124 cbr 68 [13a 2 3 800] ——- [0:0 1:0 29 2] [123] 2 0
D 30.746095199 _2_ RTR NRTE 124 cbr 68 [13a 2 3 800] ——- [0:0 1:0 28 2] [123] 2 0
...
s 33.868618722 _2_ RTR — 0 AODV 48 [0 0 0 0] ——- [2:255 -1:255 30 0] [0x2 1 2 [1 14] [2 0]] (REQUEST)
r 33.869519323 _4_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 2 [1 14] [2 0]] (REQUEST)
r 33.869519323 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [2:255 -1:255 30 0] [0x2 1 2 [1 14] [2 0]] (REQUEST)
s 33.873094625 _4_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [4:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.873994998 _5_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.873995226 _2_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.873995292 _3_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [4:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
s 33.877073906 _3_ RTR — 0 AODV 48 [0 ffffffff 2 800] ——- [3:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.877974192 _0_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.877974507 _2_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
r 33.877974572 _4_ RTR — 0 AODV 48 [0 ffffffff 3 800] ——- [3:255 -1:255 29 0] [0x2 2 2 [1 14] [2 0]] (REQUEST)
s 33.878099286 _5_ RTR — 0 AODV 48 [0 ffffffff 4 800] ——- [5:255 -1:255 28 0] [0x2 3 2 [1 14] [2 0]] (REQUEST)
r 33.879709292 _4_ RTR — 0 AODV 48 [0 ffffffff 5 800] ——- [5:255 -1:255 28 0] [0x2 3 2 [1 14] [2 0]] (REQUEST)
r 33.879709586 _1_ RTR — 0 AODV 48 [0 ffffffff 5 800] ——- [5:255 -1:255 28 0] [0x2 3 2 [1 14] [2 0]] (REQUEST)
s 33.879709586 _1_ RTR — 0 AODV 44 [0 0 0 0] ——- [1:255 2:255 30 5] [0x4 1 [1 16] 10.000000] (REPLY)
```

```
r 33.881255586 _5_ RTR --- 0 AODV 44 [13a 5 1 800] ------- [1:255 2:255 30 5] [0x4 1 [1 16] 10.000000] (REPLY)
f 33.881255586 _5_ RTR --- 0 AODV 44 [13a 5 1 800] ------- [1:255 2:255 29 4] [0x4 2 [1 16] 10.000000] (REPLY)
r 33.883604704 _4_ RTR --- 0 AODV 44 [13a 4 5 800] ------- [1:255 2:255 29 4] [0x4 2 [1 16] 10.000000] (REPLY)
f 33.883604704 _4_ RTR --- 0 AODV 44 [13a 4 5 800] ------- [1:255 2:255 28 2] [0x4 3 [1 16] 10.000000] (REPLY)
s 33.884133848 _0_ RTR --- 0 AODV 48 [0 ffffffff 3 800] ------- [0:255 -1:255 28 0] [0x2 3 2 [1 14] [2 0]] (REQUEST)
r 33.885273906 _3_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 28 0] [0x2 3 2 [1 14] [2 0]] (REQUEST)
r 33.889951741 _2_ RTR --- 0 AODV 44 [13a 2 4 800] ------- [1:255 2:255 28 2] [0x4 3 [1 16] 10.000000] (REPLY)
...
s 34.162887855 _0_ AGT --- 141 cbr 48 [0 0 0 0] ------- [0:0 1:0 32 0] [137] 0 0
r 34.162887855 _0_ RTR --- 141 cbr 48 [0 0 0 0] ------- [0:0 1:0 32 0] [137] 0 0
s 34.162887855 _0_ RTR --- 141 cbr 68 [0 0 0 0] ------- [0:0 1:0 30 3] [137] 0 0
r 34.164624715 _3_ RTR --- 141 cbr 68 [13a 3 0 800] ------- [0:0 1:0 30 3] [137] 1 0
f 34.164624715 _3_ RTR --- 141 cbr 68 [13a 3 0 800] ------- [0:0 1:0 29 2] [137] 1 0
r 34.167066518 _2_ RTR --- 141 cbr 68 [13a 2 3 800] ------- [0:0 1:0 29 2] [137] 2 0
f 34.167066518 _2_ RTR --- 141 cbr 68 [13a 2 3 800] ------- [0:0 1:0 28 4] [137] 2 0
r 34.169328321 _4_ RTR --- 141 cbr 68 [13a 4 2 800] ------- [0:0 1:0 28 4] [137] 3 0
f 34.169328321 _4_ RTR --- 141 cbr 68 [13a 4 2 800] ------- [0:0 1:0 27 5] [137] 3 0
r 34.171469439 _5_ RTR --- 141 cbr 68 [13a 5 4 800] ------- [0:0 1:0 27 5] [137] 4 0
f 34.171469439 _5_ RTR --- 141 cbr 68 [13a 5 4 800] ------- [0:0 1:0 26 1] [137] 4 0
r 34.173891439 _1_ AGT --- 141 cbr 68 [13a 1 5 800] ------- [0:0 1:0 26 1] [137] 5 0
...
```

`RREQs_AF_NI`: By `RREQ_AF_NI`, an attacker can prevent a victim node from receiving data packets for a short time. If the attacker repeats broadcasting faked RREQ message , it can isolate the victim node from receiving data packets for a long time. In the faked RREQ messages, the attacker may set the source IP address in IP header as a non-existent node, the upstream node of the broken link will launch a local repair process and attempt to establish a route to the victim node. Because this upstream node may cache the data packets from other nodes to the victim nodes, after establishing the route to the victim node, these data packets can be forwarded to the victim node. In order to isolate the victim thoroughly, the attacker can use its own IP address as the source IP address in the IP header, so it is in the routes from other nodes to the victim node. When the attacker receives data packet destined to the victim node, it just drops them. It is a kind of blackhole attack.

In the simulation, the attacker sends out 20 faked RREQ messages per second. The simulation result of `RREQs_AF_NI` is shown in Figure 7(a). `RREQs_AF_NI` is better than `RREQs_MF_NI` in isolating a node from receiving data packets. The victim node still can receive some data packets due to the random delay for a node to broadcast a RREQ message. Before the faked RREQ messages reaches other nodes, other nodes may have received RREP messages and sent out data packets to the victim node.

`RREQs_AF_RC`: An attacker may consume the resource of the network and the other nodes by broadcasting faked RREQ messages actively. The attacker may also spoof the source IP address in both the routing message and the IP header to conceal itself. In the simulation, the attacker sends 20 faked RREQ messages per second. The simulation result of `RREQs_AF_RC` is shown in Figure 3(d).

`RREPs_DR_DR`: If an inside attacker is the only node that connects two parts of an ad-hoc network, it can prevent routes between the two parts from being established by dropping the RREP messages.

RREPs_MF_RD: A source node may initiate route discovery process several times, and an inside attacker can disrupt the route by applying RREP_MF_RD several times. In the AODV protocol, intermediate nodes can send RREP messages if they have fresh enough routes to the destination nodes, so the source node may receive several RREP messages after broadcasting one RREQ message. In order to disrupt the routes, the attacker increases the destination sequence number of the receiving RREP messages and sets the source IP address in IP header to a non-existent IP address.

RREPs_MF_RI: An inside attacker can invade several routes by applying RREP_MF_RI several times. In RREP_MF_RI, to suppress other RREP messages for the same RREQ message, the attacker may increase the destination sequence number by a small number, so the source node will choose the route which goes through the attacker.

RREPs_FR_RD: Whenever the attacker receives RREQ messages from a source node to a destination node, it can disrupt the route by applying RREP_FR_RD. The simulation result is shown Figure 3(b). In another case, the attacker can cause a "Black Hole" in the ad-hoc network if she invades the routes by applying RREP_FR_RI and drops the data packets through it.

RREPs_FR_RI: Because the mobility of the mobile nodes, the route between two nodes may be lost after some time, the source node may send RREQ message again. The attacker can apply the misuse RREPs_FR_RI to keep invading the route. To achieve good result, the attacker should follow the source node in its transmission range. The simulation result is is shown in Figure 3(c).

RREPs_FR_NI: In some time, an inside attacker can isolate a victim node from sending out data packets by applying RREPs_FR_NI. Whenever the attacker receives a RREQ message originated from the victim node, it sends a faked RREP message to the victim node to disrupt the route, just as described in RREP_FR_RD. Therefore, the victim node cannot send data packets to the other nodes successfully. However, if a destination node broadcasts a RREQ message and the victim node receives this RREQ message, the victim node may have a route to the destination node. The simulation result is shown in Figure 7(b). We can see RREPs_FR_NI misuses do not always succeed.

RREPs_AF_RD: The attacker may apply RREP_AF_RD several times to disrupt a route. The simulation result is shown in Figure 3(b).

RREPs_AF_RI: The simulation result is shown in Figure 7(c).

RREPs_AF_NI: Assume an inside attack is in the transmission range of a victim node, the attacker can prevent the other nodes from receiving data packets from the victim node by sending faked RREP messages to the victim node. To achieve this goal, the attacker impersonates other nodes to send faked RREP messages to the victim node. The faked RREP messages are forged in the following way:

- Set the destintion IP address to one of the other nodes' IP address;
- Set the source IP address to the victim node's IP address;
- Increase the destination sequence number by at least one;
- Set the source IP address in the IP header to the attacker's IP address.

After receiving date packets originated from the victim node, the attacker simply drops these data packets. Here, we do not set the source IP address in the IP header to a non-existent IP address, because in such case, the upstream node which forwards data packets to the non-existent node will generate a new RREQ message or a RERR message. The attacker also needs to disrupt the route from the upstream node to the destination node. The simulation result is shown in Figure 7(b).

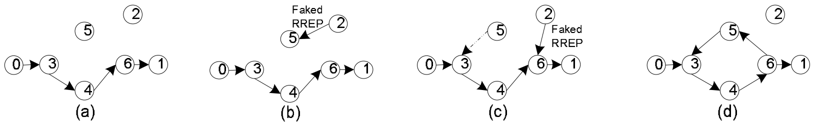


Figure 9: Form a Loop by Two Faked RREP Messages

It is rather difficult to prevent the victim node from receiving data packets from other nodes by only sending faked RREP messages, because a faked RREP message is unicasted to a neighbor of the attacker, and this neighbor node may not have a route to the destination node. However, the attacker can prevent victim node from receiving from other nodes by sending faked RREQ messages as described in `RREQs_AF_NI`.

`RREPs_AF_RC`: In `RREP_AF_RC`, the attacker can form a loop between two nodes, and in some scenarios, by sending more than one faked RREP messages, an inside attacker can form a loop containing more nodes. Suppose there is a route from node 0 to node 1 as Figure 9(a) shows. The malicious node 2 unicasts the first faked RREP message and send it to node 5, as in Figure 9(b). The attacker may forge the first RREP message as follows:

- Set the source IP address as node 0;
- Set the destination IP address as node 1;
- Set the destination sequence number to a number greater than node 1's sequence number in node 5's route table;
- Set the source IP address in the IP header as node 3;
- Set the destination IP address in the IP header as node 5.

After receiving the faked RREP message, node 5 update node 3 as the next hop to node 1. If the sequence number in the faked RREP message is greater than that in node 1's route table, a loop between node 3 and node 5 is formed. If not, the attacker may then unicast the second faked RREP message to node 6, which it may forge the second RREP message as follows:

- Set the source IP address as node 0;
- Set the destination IP address as node 1;
- Set the destination sequence number to a number greater than node 1's current sequence number;
- Set the source IP address in the IP header as node 5;
- Set the destination IP address in the IP header as node 6.

After node 6 receives this faked RREP message, it will update node 5 as the next hop to node 1. As a result, nodes 3, 4, 5, and 6 are involved in a loop, as shown in Figure 9(d).

In table 8, we can see that the attacker consumes several times more energy of the nodes in the loops than in the normal situations.

The fragment of the trace file is as follows:

...

Table 8: Power Consumption

Node	Initial Energy (Watt)	NORMAL End Energy (Watt)	RREPs_AF_RC End Energy (Watt)
Node 3	100.00	98.913347	91.848227
Node 4	100.00	98.911936	91.993235
Node 5	100.00	99.958096	92.066205
Node 6	100.00	98.912195	92.063037

s 8.634117015 _0_ AGT — 4 cbr 48 [0 0 0 0] [energy 99.982528] ——- [0:0 1:0 32 0] [4] 0 0
r 8.634117015 _0_ RTR — 4 cbr 48 [0 0 0 0] [energy 99.982528] ——- [0:0 1:0 32 0] [4] 0 0
s 8.634117015 _0_ RTR — 4 cbr 68 [0 0 0 0] [energy 99.982528] ——- [0:0 1:0 30 3] [4] 0 0
r 8.635854015 _3_ RTR — 4 cbr 68 [13a 3 0 800] [energy 99.980330] ——- [0:0 1:0 30 3] [4] 1 0
f 8.635854015 _3_ RTR — 4 cbr 68 [13a 3 0 800] [energy 99.980330] ——- [0:0 1:0 29 4] [4] 1 0
r 8.637935776 _4_ RTR — 4 cbr 68 [13a 4 3 800] [energy 99.979754] ——- [0:0 1:0 29 4] [4] 2 0
f 8.637935776 _4_ RTR — 4 cbr 68 [13a 4 3 800] [energy 99.979754] ——- [0:0 1:0 28 6] [4] 2 0
r 8.639937537 _6_ RTR — 4 cbr 68 [13a 6 4 800] [energy 99.979178] ——- [0:0 1:0 28 6] [4] 3 0
f 8.639937537 _6_ RTR — 4 cbr 68 [13a 6 4 800] [energy 99.979178] ——- [0:0 1:0 27 1] [4] 3 0
r 8.642238537 _1_ AGT — 4 cbr 68 [13a 1 6 800] [energy 99.980896] ——- [0:0 1:0 27 1] [4] 4 0
s 8.668618722 _2_ AGT — 5 cbr 48 [0 0 0 0] [energy 99.982518] ——- [2:0 4:0 32 0] [0] 0 0
r 8.668618722 _2_ RTR — 5 cbr 48 [0 0 0 0] [energy 99.982518] ——- [2:0 4:0 32 0] [0] 0 0
f 8.668618722 _2_ RTR — 0 AODV 44 [0 0 0 0] [energy 99.982518] ——- [3:255 0:255 30 5] [0x4 1 [1 2] 10.000000] (REPLY)
f 8.668618722 _2_ RTR — 0 AODV 44 [0 0 0 0] [energy 99.982518] ——- [5:255 0:255 30 6] [0x4 1 [1 20] 10.000000] (REPLY)
r 8.672759727 _6_ RTR — 0 AODV 44 [13a 6 2 800] [energy 99.976835] ——- [5:255 0:255 30 6] [0x4 1 [1 20] 10.000000] (REPLY)
f 8.672759727 _6_ RTR — 0 AODV 44 [13a 6 2 800] [energy 99.976835] ——- [5:255 0:255 29 4] [0x4 2 [1 20] 10.000000] (REPLY)
r 8.674609006 _5_ RTR — 0 AODV 44 [13a 5 2 800] [energy 99.980090] ——- [3:255 0:255 30 5] [0x4 1 [1 2] 10.000000] (REPLY)
f 8.674609006 _5_ RTR — 0 AODV 44 [13a 5 2 800] [energy 99.980090] ——- [3:255 0:255 29 3] [0x4 2 [1 2] 10.000000] (REPLY)
r 8.676679305 _4_ RTR — 0 AODV 44 [13a 4 6 800] [energy 99.975798] ——- [5:255 0:255 29 4] [0x4 2 [1 20] 10.000000] (REPLY)
f 8.676679305 _4_ RTR — 0 AODV 44 [13a 4 6 800] [energy 99.975798] ——- [5:255 0:255 28 3] [0x4 3 [1 20] 10.000000] (REPLY)
r 8.679535358 _3_ RTR — 0 AODV 44 [13a 3 5 800] [energy 99.975174] ——- [3:255 0:255 29 3] [0x4 2 [1 2] 10.000000] (REPLY)
s 8.679535358 _3_ RTR — 0 AODV 44 [13a 3 5 800] [energy 99.975174] ——- [3:255 0:255 28 0] [0x4 3 [1 2] 10.000000] (REPLY)
r 8.681544358 _0_ RTR — 0 AODV 44 [13a 0 3 800] [energy 99.976278] ——- [3:255 0:255 28 0] [0x4 3 [1 2] 10.000000] (REPLY)
r 8.684179039 _3_ RTR — 0 AODV 44 [13a 3 4 800] [energy 99.973619] ——- [5:255 0:255 28 3] [0x4 3 [1 20] 10.000000] (REPLY)
f 8.684179039 _3_ RTR — 0 AODV 44 [13a 3 4 800] [energy 99.973619] ——- [5:255 0:255 27 0] [0x4 4 [1 20] 10.000000] (REPLY)
r 8.686588039 _0_ RTR — 0 AODV 44 [13a 0 3 800] [energy 99.975059] ——- [5:255 0:255 27 0] [0x4 4 [1 20] 10.000000] (REPLY)
s 8.865538834 _0_ AGT — 6 cbr 48 [0 0 0 0] [energy 99.975059] ——- [0:0 1:0 32 0] [5] 0 0
r 8.865538834 _0_ RTR — 6 cbr 48 [0 0 0 0] [energy 99.975059] ——- [0:0 1:0 32 0] [5] 0 0

s 8.865538834 _0_ RTR — 6 cbr 68 [0 0 0 0] [energy 99.975059] ——- [0:0 1:0 30 3] [5] 0 0
r 8.867275834 _3_ RTR — 6 cbr 68 [13a 3 0 800] [energy 99.972006] ——- [0:0 1:0 30 3] [5] 1 0
f 8.867275834 _3_ RTR — 6 cbr 68 [13a 3 0 800] [energy 99.972006] ——- [0:0 1:0 29 4] [5] 1 0
r 8.869397595 _4_ RTR — 6 cbr 68 [13a 4 3 800] [energy 99.971843] ——- [0:0 1:0 29 4] [5] 2 0
f 8.869397595 _4_ RTR — 6 cbr 68 [13a 4 3 800] [energy 99.971843] ——- [0:0 1:0 28 6] [5] 2 0
r 8.871839357 _6_ RTR — 6 cbr 68 [13a 6 4 800] [energy 99.971373] ——- [0:0 1:0 28 6] [5] 3 0
f 8.871839357 _6_ RTR — 6 cbr 68 [13a 6 4 800] [energy 99.971373] ——- [0:0 1:0 27 5] [5] 3 0
r 8.876818119 _5_ RTR — 6 cbr 68 [13a 5 6 800] [energy 99.973494] ——- [0:0 1:0 27 5] [5] 4 0
f 8.876818119 _5_ RTR — 6 cbr 68 [13a 5 6 800] [energy 99.973494] ——- [0:0 1:0 26 3] [5] 4 0
r 8.878880030 _3_ RTR — 6 cbr 68 [13a 3 5 800] [energy 99.968454] ——- [0:0 1:0 26 3] [5] 5 0
f 8.878880030 _3_ RTR — 6 cbr 68 [13a 3 5 800] [energy 99.968454] ——- [0:0 1:0 25 4] [5] 5 0
r 8.881001791 _4_ RTR — 6 cbr 68 [13a 4 3 800] [energy 99.968291] ——- [0:0 1:0 25 4] [5] 6 0
f 8.881001791 _4_ RTR — 6 cbr 68 [13a 4 3 800] [energy 99.968291] ——- [0:0 1:0 24 6] [5] 6 0
r 8.883223553 _6_ RTR — 6 cbr 68 [13a 6 4 800] [energy 99.967446] ——- [0:0 1:0 24 6] [5] 7 0
f 8.883223553 _6_ RTR — 6 cbr 68 [13a 6 4 800] [energy 99.967446] ——- [0:0 1:0 23 5] [5] 7 0
r 8.885385165 _5_ RTR — 6 cbr 68 [13a 5 6 800] [energy 99.970614] ——- [0:0 1:0 23 5] [5] 8 0
f 8.885385165 _5_ RTR — 6 cbr 68 [13a 5 6 800] [energy 99.970614] ——- [0:0 1:0 22 3] [5] 8 0
...
r 99.832444195 _6_ RTR — 377 cbr 68 [13a 6 4 800] [energy 92.063037] ——- [0:0 1:0 4 6] [375] 27 0
f 99.832444195 _6_ RTR — 377 cbr 68 [13a 6 4 800] [energy 92.063037] ——- [0:0 1:0 3 5] [375] 27 0
r 99.834865808 _5_ RTR — 377 cbr 68 [13a 5 6 800] [energy 92.066205] ——- [0:0 1:0 3 5] [375] 28 0
f 99.834865808 _5_ RTR — 377 cbr 68 [13a 5 6 800] [energy 92.066205] ——- [0:0 1:0 2 3] [375] 28 0
r 99.837027718 _3_ RTR — 377 cbr 68 [13a 3 5 800] [energy 91.848227] ——- [0:0 1:0 2 3] [375] 29 0
f 99.837027718 _3_ RTR — 377 cbr 68 [13a 3 5 800] [energy 91.848227] ——- [0:0 1:0 1 4] [375] 29 0
r 99.839429480 _4_ RTR — 377 cbr 68 [13a 4 3 800] [energy 91.993235] ——- [0:0 1:0 1 4] [375] 30 0
D 99.839429480 _4_ RTR TTL 377 cbr 68 [13a 4 3 800] [energy 91.993235] ——- [0:0 1:0 0 4] [375] 30 0
....

`RERRs_AF_RD`: If the attacker is in the transmission range of a victim node, the attacker can disable all the route entries in the victim node's route table by sending out several faked RERR messages. `RERRs_AF_RD` does not require the attacker be the only neighbor of the victim node. Because the victim node may use different neighbors to different destination nodes, the attacker just impersonates all the neighbors to send a faked RERR message to the victim node. In the faked RERR message, the attacker includes all the mobile nodes in the ad-hoc networks as the unreachable destination and increases their sequence number by a small number. When the victim node receives these faked RERR messages, it will disable the routes to all other nodes.