Automated Evaluation of Secure Route Discovery in MANET Protocols

Todd R. Andel [1] * and Alec Yasinsac [2] **

1. Air Force Institute Technology, Wright-Patterson AFB, OH, USA
2. Florida State University, Tallahassee, FL, USA

Abstract. Evaluation techniques to analyze security properties in ad hoc routing protocols generally rely on manual, non-exhaustive approaches. Non-exhaustive analysis techniques may conclude a protocol is secure, while in reality the protocol may contain an unapparent or subtle flaw. Using formalized exhaustive evaluation techniques to analyze security properties increases protocol confidence. In this paper, we offer an automated evaluation process to analyze security properties in the route discovery phase for on-demand source routing protocols. Using our automated security evaluation process, we are able to produce and analyze all topologies for a given network size. The individual network topologies are fed into the SPIN model checker to exhaustively evaluate protocol abstractions against an attacker attempting to corrupt the route discovery process.

1 Introduction

Mobile ad hoc networks (MANETs) consist of portable wireless nodes that do not use fixed infrastructure. Unlike their wired counterparts that depend on fixed routers for network connectivity and message forwarding, MANETs rely on each node to provide routing functionality. Ad hoc routing protocols allow wireless nodes to communicate with nodes outside their local transmission distance. MANET routing protocols [1,2,3,4] commonly utilize two-phased routing approaches in which a route is first discovered during a route discovery phase and data is subsequently forwarded over the discovered route. In order for the protocol to meet its desired goal to deliver messages, both phases must be secured to protect the protocol against malicious activity.

There are numerous approaches proposed to analyze security properties in MANET routing protocols. These techniques include visual inspection, network simulation, analytical proofs, simulatability models, and formal methods [5]. Unfortunately, most MANET literature generally follows unstructured approaches and non-exhaustive methods to evaluate route security.

* The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

** This material is based upon work supported in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under grants numbered W91NF-04-1-0415.

In this paper, we utilize the formal methods approach through automated model checking to evaluate the ad hoc route discovery process against route corruption. Given the network topology, our analysis models exhaustively check all routing combinations to evaluate if an attacker can corrupt the route discovery phase by returning validated routes that are not consistent within the network topology. Our modeled protocol abstractions employ the SPIN model checker [6] to provide exhaustive analysis against the specified protocol security property. Although SPIN has been used to evaluate MANET routing protocol operation (e.g., loop freedom) [7,8,9], to the best of our knowledge we are the first to use SPIN to evaluate security properties in ad hoc routing protocols, as presented by our initial work in [10]. This paper extends our initial work on using SPIN to evaluate MANET route security and provides an automated process to examine all possible network topologies for a given network size.

In the remainder of this paper, we discuss requirements for secure routing and the current analysis techniques being used to evaluate secure ad hoc routing protocols. Our primary contribution is the secure routing model abstraction and the exhaustive network topology generation and analysis process. We exercise our automated security analysis process to discover an undocumented attack on the Ariadne [11] protocol and an undocumented attack on the endairA protocol [12].

2 Evaluating Secure Routing

Secure ad hoc routing protocols must be analyzed to ensure their security goals are met and to identify under what adversarial environments they may fail. We first define secure routing protocol requirements, discuss the current techniques used to evaluate MANET security properties, and provide an overview of our exhaustive analysis approach.

2.1 Requirements for Secure Routing Protocols

The term *security protocol* customarily refers to an *authentication protocol*, in which the goal is to securely share information between two nodes. Security analysis for authentication protocols evaluates if it is possible for a third party to obtain protected information, regardless of the communication path [13]. Conversely, security evaluations for MANET secure routing protocols must consider the complete communication path, or route. In particular, we must consider route accuracy and protocol reliability.

A routing protocol provides *route accuracy* if it produces routes that exist within the current network topology. Route accuracy is an integrity issue, ensuring that an attacker has not corrupted the path identified during route discovery. Since the routes obtained during route discovery can fail due to both malicious actions and non-malicious failures (e.g., mobility, hardware failures, etc.), the routing protocols must also provide *reliability*. Once a route fails, reliability mechanisms initiate a new route discovery process, use a

previously found path if multi-path protocols [14,15] are being utilized, or may attempt to detect and remove malicious nodes via probing protocols [16].

2.2 Current Evaluation Techniques

There are many techniques used to evaluate security properties in MANET routing protocols. These techniques include visual inspection, network simulation, analytical methods, simulatability models, and formal methods. Visual inspection, analytical proofs, and simulatability models are not automated. Visual inspection and network simulation do not provide exhaustive attacker analysis. Work in [5] provides detailed analysis on the capabilities and limitations of these analysis methods for use in MANET environments.

Our research follows the formal methods [17] and automated model checking [18] paradigms to evaluate route validity for routes returned in on-demand source routing protocols. Automated formal methods have shown success in evaluating authentication protocols [19,20,21]. There has also been some initial work [22,23] using automated formal methods to evaluate MANET route security for ad hoc on-demand distance vector protocols such as AODV and Secure AODV (SAODV), where the attacker's goal is to corrupt a node's next-hop entries stored in the node's routing table. However, the work was isolated to evaluating a few network topologies.

2.3 An Exhaustive Evaluation Approach

Figure 1 illustrates our complete analysis procedure. In the following sections we describe SPIN model development over the Ariadne and endairA protocols, attacker development, and our automated process that generates and analyzes all network topologies for a given network size *N*.

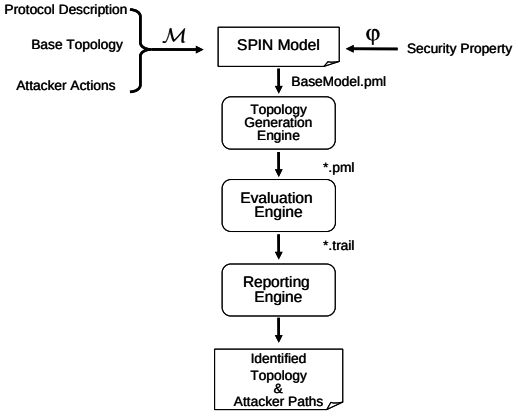


Figure 1. Automated Security Analysis Process

We transform the routing protocol, attacker actions, and security property into a model abstraction for analysis using the SPIN model checker. SPIN [6] is a general purpose model checker developed to verify the operation of distributed systems. Once a protocol abstraction and desired goals are specified in the Promela modeling language, SPIN generates a finite state automata (FSA) and performs exhaustive state analysis to determine if the desired goal holds or is violated. If the system goal is violated, SPIN generates an event sequence leading to the failure.

A primary factor in choosing SPIN for our research is its success in verifying security properties in authentication protocols [24] and its previous use to evaluate loop freedom in MANET routing protocols in non-malicious environments [7,8,9]. Our research combines these areas to analyze security properties in MANET routing protocols.

Additional factors in choosing SPIN include Promela's ability to model message passing and the ability to model independent distributed processes. These factors allow us to model the routing process in a MANET environment. Finally, Promela's ability to code non-deterministic choices allows us to model the wireless message deliverability environment. Since wireless message deliverability is not guaranteed, an attacker may be able to inject false routing information in place of dropped messages. For example, we use the following *if* construct to model wireless message reception:

```
if
 :: message received ->
            process message according to protocol
 :: message received ->
            drop message and take no actions
fi .
```

Since both entry statements are true, then either statement and its corresponding actions may be non-deterministically chosen during SPIN simulations. During SPIN exhaustive analysis, all possible non-deterministic choices are examined independently. Using SPIN in this fashion allows us to model all possible message interactions to determine if an attack may occur.

3 Model Checking Secure Routing

We focus our model abstraction on the MANET route discovery phase, analyzing if an attacker can corrupt the path during route discovery. Our model development and analysis targets source routing protocols, in which routes are explicitly embedded into protocol messages. We brake the model into the three primary processes: the wireless medium, non-malicious node, and attacker node.

For each primary process, we isolate the model development to focus on malicious failures. Non-malicious routing failures occur due to mobility or failed nodes, which affects a protocol's message deliverability and overall protocol performance. Malicious

failures may also occur, which to the routing protocol cannot be distinguished from non-malicious failures.

In addition to the inconclusive results encountered by including non-malicious failures, modeling non-malicious failures may not be feasible in a finite space model checker. For example, Wibling et al. [8] evaluate protocol loop freedom and show that it is not feasible to model mobility in model checking paradigms due to rapid state-space explosion. While model checkers may encounter the same state-space limitations when isolating malicious failures, modeling only the required failures of interest reduces the model complexity and increases the chances that the model checker can evaluate the given security property.

Based on these observations, we focus on modeling malicious failures against secure routing protocols. Eliminating non-malicious failures allows us to effectively isolate discovered route failures due to malicious activity. Any security flaws which may arise due to mobility causing an undelivered message is taken into account by exhaustively searching for all possible paths in the given topology under analysis, since transmissions in general cannot be guaranteed in wireless environments.

3.1 Modeling the Wireless Medium

Modeling message transmission in a wireless environment is a significant challenge for simulating and model checking MANET protocols. When a wireless node transmits a message in a physical MANET implementation, all nodes within its transmission footprint receive the message virtually simultaneously. Each wireless recipient then determines if it must process or drop the received packet.

SPIN does not provide direct broadcast communication support; therefore, we model the required communication components to ensure all neighbor nodes receive the message. Since time is implicitly modeled by evaluating all computational possibilities, we model one hop broadcast messages via multiple unicast messages. In [25], Ruys' discusses how to implement simple broadcast communication via a common bus structure, a matrix of channels for each node combination, or as a broadcast server process.

We model MANET communication between nodes using a *wireless medium server*, similar to Ruys' [25] broadcast server process. By modeling the wireless medium as its own process rather than modeling node-to-node channels or using a variable length bus, we limit the state-space and modularize our approach to eliminate modeling dependencies based on the network size being evaluated. We model a node's transmission range by connectivity rather than by distance, by ensuring only reachable neighbors receive transmitted messages. We store the network topology for the time instance we are evaluating in a two-dimensional array. Figure 2 illustrates the associated array for a four node network topology. The model's wireless medium process uses the array to determine the transmitting node's local neighbors and transmits messages accordingly.

The array consists of N rows by N columns, where N is the total number of non-malicious nodes plus the total number of malicious nodes. Rows indicate the transmitting node and columns indicate the node's local neighbors. Each array element holds a Boolean

value, with true (or 1) indicating a connection exists between node pairs. The shaded areas indicate array symmetry in a bidirectional environment.

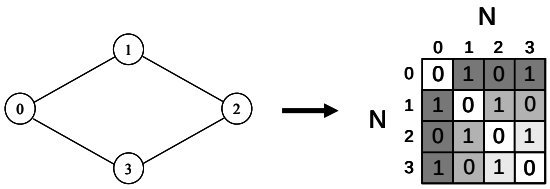


Figure 2. Representing Network Topology

Using the independent wireless medium process with the global connectivity array allows us to model any MANET routing protocol. However, we tune the wireless medium to model on-demand source routing protocols in order to maintain a reduced state-space. Our wireless medium server approach delivers broadcast route request (*rreq*) messages to each adjacent node. Each recipient node processes the message and transmits the message to the next hop using the wireless medium server. The unicast route reply (*rrep*) can use an identical process, with the exception that each node receiving the message must determine if it is included in the path before retransmitting the message to the next upstream host. In order to limit state-space, the wireless medium server only transmits to the intended recipient identified in the unicast *rrep* message and to all attacker nodes within the current transmission footprint. This approach reduces state-space yet still captures the protocol's required elements for security analysis.

3.2 Modeling Source Routing Protocols

MANET two-phased routing protocols can be classified as *proactive* or *reactive*. Proactive protocols attempt to continuously maintain fresh routes between all source-destination pairs Reactive protocols establish a route between a source and destination only when required. Reactive protocols, also known as on-demand protocols, generally use either distance vector mechanisms or source routing. In on-demand distance vector protocols, such as AODV, tables are used in each node to direct the next hop to a given destination. On-demand source routing protocols, such as DSR, explicitly embed the complete route into each message.

Our research goal is to automate the security analysis process for evaluating routing attacks against the route discovery phase in on-demand source routing protocols. In our initial work [10], we modeled the DSR protocol and the SRP extension to DSR. In this work we provide models for the Ariadne and endairA security extensions based on the DSR protocol. We make several abstraction choices to simplify the resulting model and limit the resulting state-space. Model checking over the abstraction exhaustively examines all routing possibilities for a static network. Our model abstraction does not reflect message timing issues, since message deliverability is not guaranteed in a wireless

environment. Our resultant models allow us to search for possible route violations rather than probable outcomes.

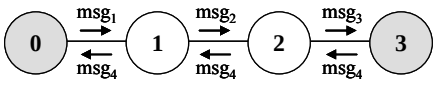
3.2.1 Modeling Ariadne

Ariadne [11] is an extension to the DSR protocol that attempts to secure the forward *rreq* from being corrupted by computing a one-way per-hop hash value at each intermediate node.

The Ariadne message formats follow as:

- *<rreq, initiator, target, id, hash_value, accum_path, sig_list>*
- *<rrep, target, initiator, accum_path, sig_list, target_sig>*.

We illustrate the Ariadne protocol using the network topology and messages shown in Figure 3, with initiator node *0*, target node *3*, *H* is a cryptographically secure one-way hash function, and SK_i is node *i's* signing key.



msg_1 = (rreq, 0, 3, id, h_0, ( ), ( ))
h_0 = H[0, initiator-target secret]
msg_2 = (rreq, 0, 3, id, h_1, (1), (sig_1))
h_1 = H[1, h_0]
sig_1 = SK_1{rreq, 0, 3, id, h_1, (1)}
msg_3 = (rreq, 0, 3, id, h_2, (1, 2), (sig_1, sig_2))
h_2 = H[2, h_1]
sig_2 = SK_2{rreq, 0, 3, id, h_2, (1, 2), (sig_1)}
msg_4 = (rrep, 3, 0, (1, 2), (sig_1, sig_2), (sig_3))
sig_3 = SK_3{rrep, 0, 3, (1, 2), (sig_1, sig_2)}

Figure 3. Ariadne Protocol Messages

The hash value (h_x) is included to guard against an attacker removing a node from the embedded path. The initial hash seed is a secret known only to the initiator-target pair. Each intermediate node adds its id to the route path *(accum_path)*, calculates and inserts a new per-hop hash value *(hash_value)*, and appends its signature before retransmitting the *rreq*. In addition to the hash value, each intermediate node computes and appends a signature over the complete packet to make sure the path contains only trusted insiders. Ariadne allows the signature to be a message authentication code (MAC) computed with a pairwise secret key, a MAC computed with a the delayed key via the TESLA broadcast key distribution scheme, or computed as a digital signature. For the purposes of our model development, we use digital signatures and refer to node *x*'s signature as sig_x.

Once the target receives the *rreq*, it validates the one-way hash value by iteratively performing the hash computation against all nodes in the accumulated path. If the *rreq* validates, the target signs the path and returns the path and signature in a *rrep*. During the *rrep*, the intermediate nodes along the unicast path relay the reply to the next upstream node indicated in the embedded path. The initiator checks the signatures and accepts the route if all checks validate.

We express Ariadne using Promela to facilitate SPIN analysis. The first step is to define the message format. The identifier (*id*) is dropped from the message since we are analyzing one route discovery round. We replace the *id* with a Boolean value in each node to ensure only one *rreq* is processed for the route discovery round. To maintain common message formatting and subsequent channel modeling, the *rreq* and *rrep* messages are modeled following the same format. The signatures of the intermediate and target nodes are combined producing the following interim message abstraction:

<*msg_type, initiator, target, accum_path, hash_chain, sig_list*>.

Ariadne does not use the *hash_chain* in the *rrep* and it is set to zero after the target processes the *rreq*. The two fields of interest are the *hash_chain* and the *sig_list*, which model the cryptographic one-way hash value and the list of cryptographic signatures respectively.

The *hash_chain* represents a cryptographically secure one-way computation. At each stage an intermediate node rehashes the value after appending its node id to the previous hash value. This process produces the one-way hash chain with *i* representing the current node and *i-1* representing the previous node along the forward *rreq*:

$$h_i = H[n_i, h_{n(i-1)}] = H[n_i, H[n_{i-1}, \ldots H[n_0, h_0] \ldots]].$$

During protocol operation the computed hash value is transmitted. For modeling purposes we remove the hash operation, listing the hash chain as: $[n_i, n_{i-1}, \ldots., n_1, n_o]$. Modeled nodes may not corrupt or remove an earlier appended value to the hash chain, since we are modeling a one-way hash value based on the chain's node identifiers in the appended array. We capture the hash computation by restricting a node's actions against the modeled hash chain to either replaying the entire chain or computing a hash value by appending to any previous hash chain captured from the wireless environment.

We represent the hash chain in Promela with an append only accumulated array. To capture the properties of the one-way hash computation, the model does not allow an intermediate node to change the *hash_chain*. The modeled hash chain is different than the accumulated path *(accum_path)*, as the model does not restrict changes to the accumulated path. Once the target receives the *rreq*, it checks the accumulated path against the hash chain to ensure they match. This process captures the effect of the target computing a one-way hash value using the path contained in the *rreq* message and comparing it against the hash value delivered in the *rreq*.

The Ariadne accumulated intermediate node signatures (*sig_list*) protect the protocol from including malicious outsiders in the routing path. The initiator validates the signatures against the received accumulated path. We model the *sig_list* in the same fashion as the hash value, appending a node id to the *sig_list* to indicate a node has signed the given message. The signatures cannot be reordered, but can be dropped in the reverse order to match any attacker dropping nodes from the end of the accumulated path. Assuming the signature process is cryptographically secure and limiting our future attack process to dropping nodes from the end of the path, we do not explicitly model the intermediate signatures in order to reduce the state-space. We do model the target signature over the path to secure the embedded path returned in the route reply. The target

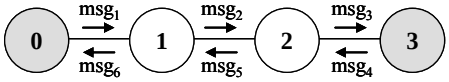
models the signature by copying the accumulated path from the forward *rreq* into the signature in the *rrep*. The signature cannot be corrupted during the *rrep*, allowing the initiator to check the returned accumulated path against the target signature to ensure routing attacks performed against the return *rrep* are detected.

The Ariadne model therefore uses the following message format, where *accum_pos* and *hash_pos* track the current position for the non-malicious node to append itself to the accumulated path and abstracted hash chain to the current forward *rreq* message. We use a single array called *crypt_val* to model both the hash chain and target signature. During the forward *rreq*, *crypt_val* holds the hash chain. During the return *rrep*, *crypt_val* holds the target signature.

<msg_type, initiator, target, accum_path, accum_pos, crypt_val, hash_pos>.

3.2.2 Modeling endairA

The endairA protocol [12] attempts to secure DSR by only securing the *rrep* using signatures, as illustrated in Figure 4. Node *0* is the initiator, node *3* is the target, and SK_i is node *i's* signing key.



msg_1 = (rreq, 0, 3, id, ( ))
msg_2 = (rreq, 0, 3, id, (1))
msg_3 = (rreq, 0, 3, id, (1, 2))
msg_4 = (rrep, 0, 3, (1, 2), (sig_3))
$sig_3 = SK_3${rrep, 0, 3, id, (1, 2), ()}
msg_5 = (rrep, 0, 3, (1, 2), (sig_3, sig_2))
$sig2 = SK_2${rrep, 0, 3, (1, 2), (sig_3)}
msg_6 = (rrep, 0, 3, (1, 2), (sig_3, sig_2, sig_1))
$sig_1 = SK_1${rrep, 0, 3, (1, 2), (sig_3, sig_2)}

Figure 4. endairA Protocol Messages

The endairA message formats follow as:

- *<rreq, initiator, target, id, accum_path>*
- *<rrep, initiator, target, accum_path, sig_list>* .

Instead of protecting the forward *rreq* process, the target computes a signature over the accumulated path received in the *rreq* and adds the signature to the *rrep*. During the *rrep*, the intermediate nodes sign the message and forward to the next hop. Once the *rrep* reaches the initiator, the initiator checks the target signature and verifies that each node in the return path has signed the message in reverse order. While the target may sign corrupted paths received by *rreq*, the protocol authors contend these paths should not make it back to the initiator with the correct appended signatures.

The approach to modeling endairA is similar to the Ariadne abstraction. As in Ariadne, we drop the *id* from the message and replace it with a Boolean value to ensure only one *rreq* is processed for the route discovery round. To maintain common message formatting

and subsequent channel modeling, the *rreq* and *rrep* messages are modeled following the same format. Our message abstraction follows as:

<msg_type, initiator, target, accum_path, accum_pos, target_sig, sig_list, sig_pos>.

The target signature (*target_sig*) holds the signature over the accumulated path that the target calculates and embeds into the *rrep*. The value is modeled by copying the *accum_path* array into the *target_sig* array. The attacker cannot change the *target_sig*, which protects the path from being corrupted during the *rrep*. The signature list (*sig_list*) keeps track of each node that has signed the *rrep* during its message traversal. We model this activity in an array, where each node adds its id to the *sig_list* during the *rrep*. Once the initiator receives the *rrep*, it checks the target signature to ensure the accumulated path matches the signature and verifies that all nodes in the accumulated path have signed the message by checking the signature list.

3.3 Modeling the Attacker

There are known attacks against MANET routing protocols that have no known detection or prevention mechanism. Attacks such as the invisible node attack (INA) [26] are a continued threat since no current mechanisms can positively identify the node that transmitted a given message. Additionally, malicious insiders can refuse to participate in routing protocols at will, even though they are trusted to follow the protocol rules.

This research focuses on an attacker that actively corrupts the embedded route during the MANET route discovery phase, resulting in routes that do not match the current network topology. The attacker model incorporates static analysis techniques to produce a finite attacker model that an automated model checker can execute. If we allowed an attacker to arbitrarily change messages, the state-space for all possible messages would quickly exceed computational capabilities. The routing messages and structure are dependent on the interactions between the intermediate nodes and the network topology, which complicates the analysis structure as compared to authentication protocol analysis.

We leverage the fact that a simplified three node (source, destination, attacker) structure can model end-to-end authentication protocol security requirements [13]. Rather than using model checking to generate all possible message structures, we evaluate all possible path sequences through the wireless network structure. To develop the attacker model we follow the approach in [24] to limit the attacker's actions based on the information provided by the protocol messages that could possibly enable an attack. This process requires static analysis over the possible messages the attacker can capture and extracts only the information required by the attacker to perform a successful attack. The information obtained by the static analysis allows the attacker to model a finite set of attempted attack sequences.

3.3.1 Ariadne Attack Development

We use the messages in Figure 3 for static analysis to develop the Ariadne attacker. Since the target node signs the accumulated path received in the *rreq*, the path cannot be corrupted during the *rrep*. Thus, the target signature limits the attacker to the forward *rreq*. Examining the *rreq* message structure indicates that the unprotected accumulated path can be changed as long as the corresponding hash chain and signatures match the path changes. The forward intermediate signatures do not allow the attacker to reorder the path or add an outsider to the path. Since we are not explicitly modeling the forward signatures, we limit the malicious insider to only appending or dropping a node from the end of the accumulated path. We also limit the malicious outsider to only dropping a node from the end of the accumulated path, since the outsider cannot generate a valid signature without a trusted key.

The final message element to consider is the one-way hash value. Due to its cryptographic properties, the hash value cannot be directly corrupted. Any attacker that attempts to drop a node from the accumulated path must compute a hash value that corresponds to the new path. The only way for an attacker to strip a node from the end of the accumulated path and compute a correct corresponding hash value is to generate the hash value based on capturing a previous hash value from an upstream neighbor during the current route discovery round. Hash values from previous route discovery rounds would not match due to the messages id or sequence number. For instance, if an attacker wishes to remove node 2 from the protocol example in Figure 3, it needs to know the value h_1. If h_1 is not captured from msg_2 it can be calculated from h_0 (captured in msg_1), since $h_1 = H[1,h_0]$. The resulting *malicious insider* attacker process proceeds as follows. Upon receiving a *rreq* from an intermediate node, the attacker stores the associated node's hash value and checks to see if it knows a hash value from a previous upstream neighbor. If the attacker has this knowledge, it drops the last node and adds its id to the accumulated path The attacker also replaces the hash value with its local hash calculation. To compute a valid hash the modeled attacker appends nodes to a previous hash-chain until it matches the current accumulated path. During the *rrep*, the attacker attempts to relay the *rrep* to any upstream neighbor. The *malicious outsider* follows the same process except that it does not add its id to the path after dropping the last node from the accumulated path.

These attacks are possible as long the attacker can capture a previous enabling hash value and the *rrep* is deliverable to one of the attacker's upstream neighbors.

3.3.2 endairA Attack Development

We use the messages in Figure 4 for static analysis to develop the endairA attacker. As in Ariadne, the target signature on the accumulated path ensures the path cannot be corrupted during the *rrep*. Since the signatures added by the intermediate nodes during the *rrep* must match the reverse order as the path signed by the target in the *rrep*, any attack must be limited to malicious insiders. If a malicious node strips a node during the forward

rreq, it requires a direct link to an upstream node to ensure the next node can produce the appropriate signature during the *rrep*. However, this link would constitute a valid path. The only avenue that allows the malicious insider the ability to drop nodes without this direct link is the ability to generate signatures for more than one node. If we assume the cryptographic process is secure in polynomial time, the only way the attacker can produce multiple signatures is to have multiple keys. In a colluding environment, we assume that all colluding attackers have previously shared their keying material.

The resulting attacker model in a colluding environment proceeds as follows. If the forward *rreq* passes through two attackers, the second attacker strips the nodes between the two attackers before sending the *rreq* to the next hop. Once the target signs the corrupted accumulated path, it responds with the *rrep*. During the *rrep* the second attacker signs for both attackers in the correct order to ensure the signatures match the nodes in the accumulated path. The requirement to enable this attack is that the second attacker must be able to relay information to the upstream node prior to the first attacker.

4 Automated Topology Generation and Analysis

The protocol and attacker models allow us to use SPIN to examine if a message sequence exists that allows an attack for the given network topology being analyzed. In [10] we chose an enabling attacker topology based on static analysis; however, a complete automated analysis capability should not rely on manually choosing an enabling topology. One of the biggest impediments to manual analysis methods is the intuition to choose a network configuration in which the attack exists.

To solve this problem, we use our SPIN models to evaluate all network topologies for *N* number of nodes. The evaluation process consists of specifying *N* and evaluating the model against each possible topology. The process relies on duplicating the Promela file for each topology, adapting the network connectivity array to reflect each configuration.

The network and corresponding connectivity array in Figure 2 represent a symmetric (i.e., bi-directional) network. The shaded areas indicate which array portions are symmetric. If modeling an asymmetric (i.e., directional) network, the symmetric areas within the connectivity array do not exist. In the asymmetric case the number of topologies is determined by:

$$\underbrace{2^{(N-1)} \times 2^{(N-1)} \times \ldots \times 2^{(N-1)}}_{N} = 2^{N(N-1)}.$$

When modeling symmetric networks, the number of topologies is based on the possibilities for the shaded areas to one side of the zero diagonal. Since the number of elements that determine the symmetric topology are half of the asymmetric case, we can calculate the number of symmetric topologies by: $2^{N(N-1)/2}$.

In this research, we analyzed the modeled protocols using a 5-node symmetric network. With $N = 5$, a total of 1,024 unique topologies are possible. Following the complete

analysis process shown in Figure 1, the remaining steps consist of the Topology Generation Engine, the Evaluation Engine, and the Reporting Engine. These steps utilize Perl routines to generate, evaluate, and report on the attacker capability for all possible network topologies for a given network size *N*.

The Topology Generation Engine generates a new Promela file for each possible network topology for the given network size *N*. The 5-node symmetric topology array can be illustrated according to Figure 5.

	0	1	2	3	4
0	0	val-9	val-8	val-7	val-6
1	val-9	0	val-5	val-4	val-3
2	val-8	val-5	0	val-2	val-1
3	val-7	val-4	val-2	0	val-0
4	val-6	val-3	val-1	val-0	0

Figure 5. 5-Node Symmetric Topology Array Values

By listing the network connectivity array values in this manner, we can represent the upper (or lower) half of the array with a binary string value as illustrated in Figure 6. We iterate the binary string through all possible values to generate all possible topologies, starting with the base topology that stores the initial network as fully disconnected (i.e., all links set to 0). Since $N = 5$, the binary string is enumerated between 0 and 1023 to represent all possible topologies. For each network topology we read the base Promela file and adjust the network connectivity array as appropriate for the current binary string representation, saving the output as *filename_case-#.pml*.

val-9	val-8	val-7	val-6	val-5	val-4	val-3	val-2	val-1	val-0

Figure 6. Binary String Representation

The next step in the analysis process is the Evaluation Engine, which compiles and executes each Promela file for SPIN exhaustive analysis. Any successful attacks are stored in individual SPIN trail files that capture the attack event sequence. The final step occurs in the Reporting Engine, which reads each error file and associated Promela model file to report the discovered attacks. The information returned is the initiator, target, and attacker nodes, along with the corrupted path and the current network topology.

5 Evaluation Criteria and Results

In addition to modeling the protocol and attacker, we define the desired security property (φ) as:

φ = all returned routes must exist in the current network topology.

To evaluate the property in SPIN we add an analysis check for the path once it is received and accepted by the node that initiated the route discovery process. The node evaluates the returned path to ensure each link exists in the model's connectivity array. If any link check fails, an assertion violation is raised and SPIN halts execution, creating a trail file that lists the event sequence leading to the failure.

Our automated process ensures all topologies for a given network size *N* are generated and subsequently evaluate using SPIN. The following analysis assumes that *N=5*.

5.1 Attacking Ariadne

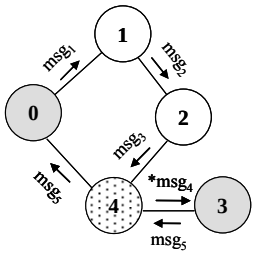
Running the analysis procedure against Ariadne and a malicious insider produces the following attack output as generated by the Report Engine:

```
Processing case: ari_nda-rreq_5node_case-611.pml
Initiator Node: 0       Target Node: 3       Attacker Node: 4       Corrupted Path: 0 - 1 - 4 - 3
The topology is:
    net_con[0].to[0] = 0 net_con[0].to[1] = 1 net_con[0].to[2] = 0 net_con[0].to[3] = 0 net_con[0].to[4] = 1
    net_con[1].to[0] = 1 net_con[1].to[1] = 0 net_con[1].to[2] = 1 net_con[1].to[3] = 0 net_con[1].to[4] = 0
    net_con[2].to[0] = 0 net_con[2].to[1] = 1 net_con[2].to[2] = 0 net_con[2].to[3] = 0 net_con[2].to[4] = 1
    net_con[3].to[0] = 0 net_con[3].to[1] = 0 net_con[3].to[2] = 0 net_con[3].to[3] = 0 net_con[3].to[4] = 1
    net_con[4].to[0] = 1 net_con[4].to[1] = 0 net_con[4].to[2] = 1 net_con[4].to[3] = 1 net_con[4].to[4] = 0
```

Further investigation into the associated SPIN trail file shows the message sequence leading to the route corruption attack. Figure 7 illustrates the message sequence for the identified topology that returns the corrupted route. Recall we are not explicitly modeling the signatures; however, the attacker actions from the trial file result in the complete attack. The attacker node removed node 2 from msg_3, added itself to the current accumulated path, computed the matching hash value, and transmitted the corrupted path in msg_4. The resulting corrupt path is returned in msg_5 as *0-1-4-3*. This attack violates both the security property φ and an original Ariadne security claim. In [11], the Ariadne authors claim that a single malicious insider cannot remove a node during route discovery. This attack was also previously indicated in [12] using the simulatability method and subsequently reported using visual inspection techniques. This research demonstrates the ability to automatically discover the attack through model checking techniques.

Analysis over Ariadne against a malicious outsider reports a similar attack, with the exception that the attacker node 4 does not add itself to the accumulated path in msg_4. The resulting corrupt path is *0-1-3*. This attack violates both the security property φ and an original Ariadne security claim. In [11], the Ariadne authors claim an attacker with no comprised keys and any number of attacker nodes, can only perform a wormhole attack or force the protocol to choose an attacker desired path by rushing, which occurs when the attacker node responds to the route discovery process faster than the protocol expects. We have shown how a malicious outsider can actively change the embedded routing path. This attack is the first to our knowledge that shows a malicious outsider can actively corrupt the Ariadne route discovery process. In order to drop a node from the accumulated

path during the *rreq*, the attacker requires only the ability to generate the appropriate hash value, based on capturing a hash from an upstream node in the path. The attacker also must be able to relay the return *rrep* to any upstream neighbor in the unicast *rrep*.



msg_1 = (rreq, 0, 3, id, h_0, ( ), ( ))
h_0 = H[0, initiator-target secret]
msg_2 = (rreq, 0, 3, id, h_1, (1), (sig_1))
h_1 = H[1, h_0]
sig_1 = SK_1{rreq, 0, 3, id, h_1, (1)}
msg_3 = (rreq, 0, 3, id, h_2, (1, 2), (sig_1, sig_2))
h_2 = H[2, h_1]
sig_2 = SK_2{rreq, 0, 3, id, h_2, (1, 2), (sig_1)}
*msg_4 = (rreq, 0, 3, id, h_4, (1, 4), (sig_1, sig_4))
h_4 = H[4, h_1] = H[4, H[1, h_0]]
sig_4 = SK_4{rreq, 0, 3, id, h_4, (1, 4), (sig_1)}
msg_5 = (rreq, 0, 3, (1, 4), (sig_1, sig_4), (sig_3))
sig_3 = SK_3{rreq, 0, 3, id, h_4, (1, 4), (sig_1, sig_4)}

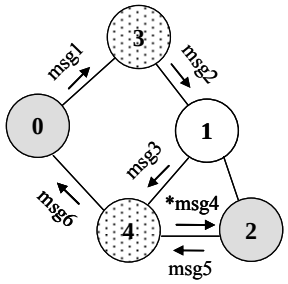
Figure 7. Ariadne Attack Sequence

5.2 Attacking endairA

Running the analysis procedure against endairA and two colluding malicious insiders produces the following attack output as generated by the Report Engine:

```
Processing case: end_nda-2_5nodel_case-250.pml
Initiator Node: 0       Target Node: 2       Attacker Node: 4       Corrupted Path: 0 - 3 - 4 - 2
The topology is:
    net_con[0].to[0] = 0 net_con[0].to[1] = 0 net_con[0].to[2] = 0 net_con[0].to[3] = 1 net_con[0].to[4] = 1
    net_con[1].to[0] = 0 net_con[1].to[1] = 0 net_con[1].to[2] = 1 net_con[1].to[3] = 1 net_con[1].to[4] = 1
    net_con[2].to[0] = 0 net_con[2].to[1] = 1 net_con[2].to[2] = 0 net_con[2].to[3] = 0 net_con[2].to[4] = 1
    net_con[3].to[0] = 1 net_con[3].to[1] = 1 net_con[3].to[2] = 0 net_con[3].to[3] = 0 net_con[3].to[4] = 0
    net_con[4].to[0] = 1 net_con[4].to[1] = 1 net_con[4].to[2] = 1 net_con[4].to[3] = 0 net_con[4].to[4] = 0
```

Figure 8 illustrates the attacker message sequence revealed in the associated SPIN trail file.



msg_1 = (rreq, 0, 2, id, ( ))
msg_2 = (rreq, 0, 2, id, (3))
msg_3 = (rreq, 0, 2, id, (3, 1))
*msg_4 = (rreq, 0, 2, (3, 4))
Node 4 drops node 1
msg_5 = (rrep, 0, 2, (3, 4), (sig_2)
sig_2 = SK_2{rrep, 0, 2, (3, 4) }
msg_6 = (rrep, 0, 2, (3, 4), (sig_2, sig_4, sig_3))
sig_4 = SK_4{rrep, 0, 2, (3, 4), (sig_2)}
sig_3 = SK_3{rrep, 0, 2, (3, 4), (sig_2, sig_4)}

Figure 8. endairA Attack Sequence

To the best of our knowledge, we provide the first active route corruption attack against endairA performed by two malicious insiders. During the *rreq*, node 4 removes the intermediate node between itself and the colluding attacker node 3. During the *rrep*, node 4 signs for both itself and node 3 since the colluding nodes share their cryptographic keys. The initiator believes msg_6 is from node 3, since the signatures are correct and the initiator cannot physically identify that node 4 actually sent the message. The resulting corrupt path is *0-3-4-2*. This attack violates both the specified security property φ and an original endairA security claim. In [12], the authors claim that endairA cannot be attack by colluding adversaries unless the adversaries are local neighbors to one another.

6 Conclusion

In this paper, we provide an automated model checking technique to evaluate route corruption attacks against the route discovery phase for on-demand source routing protocols. The existing security analysis techniques used to evaluate security properties in MANET routing protocols are not automated or do not provide exhaustive attacker analysis.

Our automated analysis process uses the SPIN model checker to examine all message event sequences for a given topology. We additionally provide exhaustive topology generation to ensure all possible network topologies for a given network size N are evaluated. Each topology is subsequently analyzed with SPIN, identifying any configuration and corresponding event sequence producing attacks along with the attack sequence.

Through the use or our automated evaluation process we identified previously undocumented attacks against the Ariadne and endairA protocol and have shown the feasibility of using model checking to automate attack analysis in MANET routing protocols.

Our future work includes introducing more detail into the protocol models. As we weigh the low level protocol details against the required state-space, our goal is to develop a protocol entirely in the model checking paradigm to meets its goals and provide subsequent compilation to produce executable routing code for network devices.

References

[1] Royer, E.M., Toh, C.-K.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications **6** (1999) 46-55

[2] Hu, Y.C., Perrig, A.: A survey of secure wireless ad hoc routing. IEEE Security & Privacy **2** (2004) 28-39

[3] Argyroudis, P.G., O'Mahony, D.: Secure routing for mobile ad hoc networks. IEEE Communications Surveys & Tutorials **7** (2005) 2-21
[4] Djenouri, D., Khelladi, L., Badache, A.N.: A survey of security issues in mobile ad hoc and sensor networks. IEEE Communications Surveys & Tutorials **7** (2005) 2-28
[5] Andel, T.R., Yasinsac, A.: Surveying Security Analysis Techniques in MANET Routing Protocols. IEEE Communications Surveys & Tutorials **9** (2007) 70-84
[6] Holzmann, G.J.: The model checker SPIN. IEEE Transactions on Software Engineering **23** (1997) 279-295
[7] De Renesse, F., Aghvami, A.H.: Formal verification of ad-hoc routing protocols using SPIN model checker. 12th IEEE Mediterranean Electrotechnical Conference, Vol. 3 (2004) 1177-1182
[8] Wibling, O., Parrow, J., Pears, A.: Automatized Verification of Ad Hoc Routing Protocols. Formal Techniques for Networked and Distributed Systems (FORTE 2004), Vol. 3235. Springer-Verlag (2004) 343-358
[9] Bhargavan, K., Obradovic, D., Gunter, C.A.: Formal verification of standards for distance vector routing protocols. Journal of the ACM **49** (2002) 538-576
[10] Andel, T.R., Yasinsac, A.: Automated Security Analysis of Ad Hoc Routing Protocols Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07), Wroclaw, Poland (2007) 9-26
[11] Hu, Y.-C., Perrig, A., Johnson, D.B.: Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks Wireless Networks **11** (2005) 21-38
[12] Ács, G., Buttyán, L., Vajda, I.: Provably secure on-demand source routing in mobile ad hoc networks. IEEE Transactions on Mobile Computing **5** (2006) 1533-1546
[13] Ryan, P., Schneider, S.: Modelling and Analysis of Security Protocols. Addison-Wesley, Harlow, England (2001)
[14] Burmester, M., Van Le, T.: Secure multipath communication in mobile ad hoc networks. International Conference on Information Technology: Coding and Computing (ITCC '04), Vol. 2 (2004) 405-409
[15] Kotzanikolaou, P., Mavropodi, R., Douligeris, C.: Secure Multipath Routing for Mobile Ad Hoc Networks. Second Annual Conference on Wireless On-demand Network Systems and Services (WONS '05) (2005) 89-96
[16] Awerbuch, B., Holmer, D., Nita-Rotaru, C., Rubens, H.: An on-demand secure routing protocol resilient to byzantine failures. 3rd ACM Workshop on Wireless Security. ACM Press, Atlanta, GA, USA (2002) 21-30
[17] Clarke, E.M., Wing, J.M.: Formal methods: state of the art and future directions. ACM Computing Surveys **28** (1996) 626-643
[18] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA (1999)
[19] Meadows, C.: The NRL Protocol Analyzer: An Overview. The Journal of Logic Programming **26** (1996) 113-131

[20] Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Tools and Algorithms for the Construction and Analysis of Systems, Vol. 1055. LNCS (1996) 147-166
[21] Song, D., Berezin, S., Perrig, A.: Athena: A novel approach to efficient automatic security protocol analysis. Journal of Computer Security **9** (2001) 47-74
[22] Nanz, S., Hankin, C.: A framework for security analysis of mobile wireless networks. Theoretical Computer Science **367** (2006) 203-227
[23] Yang, S., Baras, J.S.: Modeling vulnerabilities of ad hoc routing protocols. 1st ACM Workshop on Security of Ad hoc and Sensor Networks (2003) 12-20
[24] Maggi, P., Sisto, R.: Using SPIN to Verify Security Properties of Cryptographic Protocols. 9th international SPIN Workshop on Model Checking of Software, Vol. 2318. Springer-Verlag (2002) 187-204
[25] Ruys, T.C.: Towards effective model checking. Department of Computer Science. University of Twente, Deventer, The Netherlands (2001)
[26] Andel, T.R., Yasinsac, A.: The Invisible Node Attack Revisited. 2007 IEEE SoutheastCon, Richmond, VA (2007) 686-691