An Extensible Environment for Evaluating Secure MANET

Yongguang Zhang
HRL Laboratories, LLC
ygz@hrl.com

Yi-an Huang
Georgia Institute of Technology
yian@cc.gatech.edu

Wenke Lee
Georgia Institute of Technology
wenke@cc.gatech.edu

Abstract

Developing and evaluating secure MANET (mobile ad-hoc networks) in real systems is a complex process that involves careful design of attack test cases and security countermeasures, as well as meaningful performance measurements to evaluate both the impact of attacks and the performance of security solutions. It is desirable to have a development and testing environment that can automate this process. In this paper, we propose a software framework for such an environment and describe a system implementation in the secure MANET routing domain. This environment includes the following three major features. First, the environment is built upon a wireless network emulation tool to support repeatable experimentation. Second, it adds an attack emulation layer with necessary API for easy development and execution of attack test cases. Third, the extensible attack library includes a full set of basic attacks at its core and a way to compose complex attacks from the atomic elements. To demonstrate the usefulness of this tool, we show the development of an Intrusion Detection System (IDS) as a case study. Our successful experience confirms that the platform can greatly facilitate the development of security solutions on MANET.

1. Introduction

The history of security research and practice has taught us that security is an on-going process and any secure system should undergo rigid test and re-test with carefully designed attack test cases. Securing Mobile Ad-hoc Networks (MANET) should also follow this process and it is extremely important to evaluate secure MANET software in real systems and under real attacks.

Like any security system, a thorough evaluation of secure MANET software requires a cycle of four steps (Figure 1). First, we must understand application objectives because the ultimate test of success for a secure MANET is how well the MANET application achieves its designed mission goal in spite of threats and attacks. This *application understanding* will help us design experiments, including the choice of test cases and evaluation models. In *test case development*, we need to come up with a set of carefully designed attack scenarios. Much like other testing in software engineering disciplines, this should come after an extensive analysis of the potential threats to MANET objectives and the set of test cases should cover these threats extensively. Systematic approaches like *attack taxonomy* [7] can be used here in secure MANET test case development.

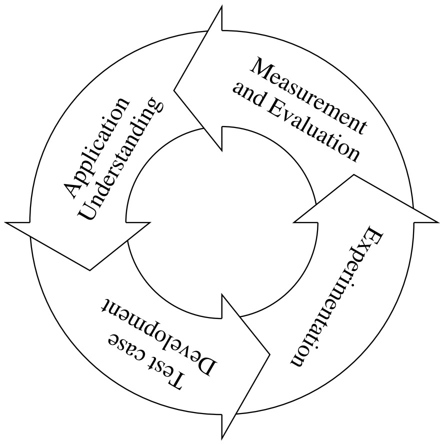

Figure 1. The Circle of Securing MANET

Next, secure MANET should be evaluated through *experimentation*. Unlike simulation, actual experiments can allow both the actual application and security codes to run in the same condition as in actual deployment. When the attack test cases are injected into the experiment to create real intrusions, the behavior of the secure MANET system can be observed and studied in face of these attacks. Furthermore, meaningful *measurement and evaluation* can be conducted to gain qualitative and quantitative assessments.

Our past experience has also taught us that this process is non-trivial and time-consuming, and it is desirable to have software tools and environments to automate and facilitate some of the tasks. Although there are some such tools available for wired networks (such as LARIAT [11, 12]), little work has been done in the wireless domains and to the best of our knowledge no such secure MANET testing systems exist in the open literature.

The goal of this research is to develop such environment as an experimentation platform for evaluating secure MANET. Given the potentially large amount of test cases and MANET scenarios, this platform should support reproducible experiments and posses the ability to inject attacks (test cases) automatically during an experiment. Further, this platform should provide easy programming support for test case development, and a way to organize attack test cases in an extensible repository.

We have developed a software system that meets the above goal. In the rest of this paper, we will describe the software architecture and explain each major components in details. Especially, we will focus on the methodology and practice of attack emulation in Section 4 and 5. As a case study, we have used this environment in our research, specifically in the development of an Intrusion Detection System (IDS) for MANET. We will report this experience in Section 7.

2. Architecture

We have developed such a software platform to facilitate the security development and evaluation that meets the above goal. Architecture-wise it includes the following components:

- *A network emulator to provide a high-fidelity communication environment for repeatable and scalable mobile ad-hoc network experiments.* This will allow us to test real security code in real applications and real systems. The emulation of underlying communication environment will allow us to test security in a wide range of different scenarios and mobility patterns.

- *An attack emulation system that is capable of injecting attack test cases during the experiments.* It installs hooks in certain MANET components and provides a programming abstraction (an API) so that researchers can write attack logics in a way independent from the actual MANET implementations. This is particularly useful because there can be a large number of test cases and it is impractical to modify a huge number of MANET components to implement each test case.

- *An extensible repository for test cases.* The attack library should have the structure to organize all the test cases and make it easily extensible to accommodate future attacks. Based on the results of attack taxonomy study [7], all attacks in MANET can be composed from a set of *basic attacks* or other compound attacks. We should therefore provide an object-oriented hierarchy in the repository to organize the test cases, to assist attack composition into more complex attacks, and to make it easy to add new atomic or compound attacks.

- *A collection of instrumental, measurement, and data analysis tools* to measure the effectiveness and performance of the security solution in the experiments. This includes tools to log traffic and security-related events, tools to observe the state of the network, and finally, tools to assess the effectiveness of the attacks and the state of the applications.

2.1. Rationale for the Emulation Approach

The evaluation of secure MANET must be under a realistic MANET environment. Although simulation tools like ns-2 [1] and QualNet is widely used for other MANET related experiments, they are not very suitable for this purpose. First, they do not have real applications and thus attacks on application level cannot be easily ported and evaluated. Second, it is impractical to obtain real and meaningful measurement data in a simulation platform. And most importantly, the security of a real system should only be evaluated with the real system and not on a simulated one.

The experimental environment should also be reproducible because this is important in exploring design space and evaluating alternatives. A full-blown test with real wireless hardware may not be repeatable because it is difficult to reproduce the extra wireless communication environment and it can be too costly to try a wide range of mobility scenarios [15]. Comparatively, the emulation approach has the advantage of both. We therefore believe that emulation is the right approach for experiment with real applications and real systems and yet be faithful to the actual communication environment (MANET).

2.2. S-MobiEmu

We have implemented this platform in a software system called S-MobiEmu, using ad-hoc routing as an example application domain and intrusion detection as a security solution case study. Figure 2 illustrates the software components and their relationship with respect to the security solution being studied.

To support reproducible experiments, we build upon a publically available wireless network emulator called MobiEmu [15]. We add an attack emulation layer, called *Basic Ad-hoc Security Routines* (BASR), as a common abstraction layer for attack injection and for test case development. The test case repository is implemented as an attack library, which extends from a core foundation library consisting of a full set of basic attacks. The repository is extensible as complex attacks can be constructed using existing attacks as building blocks. Initially, we have included several such

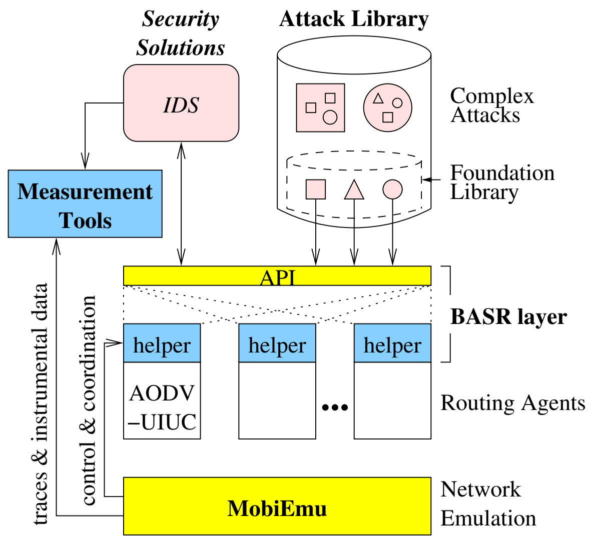

Figure 2. S-MobiEmu Software Architecture

well-known complex attack scenarios. Finally, a set of measurement tools are also provided in a performance measurement toolkit. Since functionwise this can be considered as an extenstion to MobiEmu, we call it S-MobiEmu, with "S" meaning security.

3. Emulating MANET

The network emulation system in S-MobiEmu is based on MobiEmu [15] – a software tool for testing "live" MANET systems in a laboratory setting. MobiEmu uses a cluster of n linux machines to emulate a MANET of n nodes (see Figure 3). Although these testbed hosts are physically well-connected, the packet delivery behavior has been modified at the network device layer to generate the effect of real-world wireless communications and network dynamics. With MobiEmu, MANET software can be tested under the same wireless communication characteristics and networking environment as if it were running in a real MANET deployment.

MobiEmu experiments are driven by predefined *network scenario*, which is expressed in a history of node motions and link characteristics changes. The node motions can determine current connectivity topology, and the link characteristics include bandwidth, delay, bit-error-rate, and loss rate. MobiEmu software enforces the topology and link characteristics by setting proper packet filtering and queuing rules at the device driver layer.

The MobiEmu system operates in a master/slave architecture. The master controller runs at a dedicated host outside the testbed network; a slave controller runs at each testbed host. The master controller controls all slaves and synchronizes their actions: the master dictates when

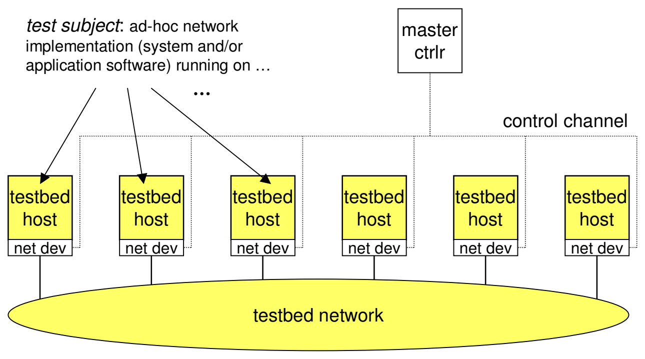
test subject: ad-hoc network implementation (system and/or application software) running on ...
...
master ctrlr
control channel
testbed host
net dev
testbed network

Figure 3. Emulating MANET with MobiEmu

changes (to topology and link characteristics) are needed according to the scenario and instructs the slaves to enforce such changes. The master/slave communication is on a separate control channel, which may be overlay on the testbed network if the overall load is low.

There are many benefits of using MobiEmu to evaluate secure MANET. First, all networking and above is real in MobiEmu, meeting our requirements to run experiments with actual secure MANET code. Second, since the communication effects are emulated, these experiments are reproducible. And third, since MobiEmu allows we run a wide range of MANET scenario without the need to physically move the nodes, we can easily repeat the experiments for a large set of test cases. Our experience of using MobiEmu in secure MANET research has further validated these points.

We have therefore use MobiEmu as the basis of our S-MobiEmu platform. To run experiments, secure MANET software (i.e., the test subject) will be loaded in each testbed host and run as if it were in a real deployment. S-MobiEmu accepts all MobiEmu scenarios, although not all attack test cases would make sense in all network scenarios. MobiEmu master controller has been extended to control and coordinate with attack emulation so that both network emulation and attack emulation are in sync.

4. Attack Emulation

The attack emulation layer in S-MobiEmu provides the means to test the security aspect of the MANET system running in MobiEmu. It interacts with each node's software stack to inject the effects of a network under attacks. For example, if a MANET attack aims at compromising a node's routing agent and falsifying its route table, the attack emulation layer will instruct the routing agent to make such alteration in its route table as if it were indeed compromised. Then, the whole system can be put under test to see how it responds to such route alteration event.

4.1. The BASR Layer

The software layer that implements the attack emulation layer is called BASR (Basic Ad-hoc Security Routines). It consists of a set of "helper" modules that implement a library of convenient security routines and a common API for attack test cases (see Figure 2). Currently, BASR is designed for ad-hoc routing although it is extensible to support other application domains.

The purpose of BASR is to isolate the implementation of attack test cases and security systems from the routing protocol code as much as possible. In real network security scenarios, routing agents can be compromised and driven into running malicious codes. Implementing attacks or countermeasures to such attacks often requires modification of the routing agents. It is obviously inconvenient and error-prone to modify the routing agents every time a new possible attack is studied. Instead, the BASR layer abstracts the most common security routines into a common API to expedite the design of attack cases and security systems, thus minimizing direct code-injection into the routing agents.

The implementation of these "helper" modules is obviously routing protocol and implementation dependent. It is indeed necessary to modify route agent source code to implement the security functions provided by the API. That is, each instance of routing protocol implementation should be paired with an instance of the helper module as illustrated in Figure 2. So far, we have implemented a BASR instance for AODV-UIUC [9], a public implementation of the AODV routing protocol. Further implementation on other protocols, such as DSR, is currently under development.

4.2. API Details

BASR supports the following three types of common routines:

1. Capturing and intercepting incoming and outgoing packets – the pcap [8] library is used to capture network packets, including both data packets and routing messages.

2. Overhearing traffic in neighboring nodes – wireless interface is put in the promiscuous mode to monitor traffic in the proximity of this node.

3. Access to routing table entries – routing table entries that usually reside internally to routing agents are made available in a shared memory block.

Here are the function prototypes of these common APIs:

-
  ```
  register_callback(bool incoming, int type,
      addr_t src, addr_t dst, func callback);
  ```

  This function creates a packet-matching rule and associates it with the given callback function. The callback function will be called, when a packet is received (when `incoming` value is true) or sent (when `incoming` value is false) at the wireless interface matches the given source, destination and protocol type. Protocol type can be specified as `TCP`, `UDP`, `RREQ`, `RREP`, `RERR`, etc., or bitwise-ORs of them, such as `TCP|UDP`. Source or destination address can be any IP address or wildcards. The callback function has the following form:

  ```
  int callback(addr_t src, addr_t dst,
      void * data, int len);
  ```

  Sequential calls of this API (possibly from different processes) will register a chain of callback functions that will be invoked in the reversed order of registrations.

-
  ```
  register_overhear_callback(int type,
      addr_t src, addr_t dst, func callback);
  ```

  This function registers a callback function similar to the previous one, but it matches only those packets that are overheard in the neighborhood. The callback function takes an additional parameter that specifies the particular neighbor from which the packet is overheard.

-
  ```
  rentry * read_route_entry(int dst);
  write_route_entry(int dst, rentry * new_rentry);
  ```

  They provide read and write access to the routing table entry (rentry) corresponding to the given destination. The rentry structure includes fields essential to the routing protocol, such as destination, next hop (or source route), hops and sequence number, etc.

-
  ```
  rentry * read_local_entry();
  write_local_entry(rentry * new_rentry);
  ```

  They provide the interface to read and modify information of the host node itself. The interfaces are similar to the `read_route_entry` and `write_route_entry`.

4.3. An Example Attack Written in the API

We now use a simple example to demonstrate how we can use the API to program attack test cases. Let us assume a possible attack scenario: an attacker Malice tries to eavesdrop in communication from Alice to Bob. Let us assume they reside on nodes M, A and B respectively. Malice can achieve the goal by several means. The simplest approach (Approach I) is to intercept all traffic from A to B on the local interface of M. It only works when M is in the route path from A to B. The second approach (Approach II) improves by overhearing nearby traffic as well. It works when there is at least some of M's neighbors resides in the interested route path. The most aggressive approach (Approach III) tries to proactively advertise a new route from A to B that contains M. Thus, no matter where Malice resides, it may always intercept the expected communication. Section 5.2 will describe the detailed technique to advertise a false route, briefly, a *Route Request* message is fabricated and it contains falsified originator and target fields, namely, B and A. By manipulating sequence number fields in the message, all nodes who receive the request will forward the message to other nodes. As a side effect, they will also update their route to B (the originator) via M (the previous hop). Eventually, A will also receive the message and update the route path to B accordingly, which contains M.

The following pseudo code segment illustrates how we can implement these three approaches with the BASR library. We assume that `disclose_data()` is a callback function that attempts to extract useful information from an intercepted data packet, and the function `broadcast()` broadcasts a packet. Here we provide a simplified RREQ structure only for demonstration purposes.

```
Eavesdrop_Approach_I(addr_t A, addr_t B)
{
    BASR::register_callback(true, TCP|UDP, A, B,
        disclose_data);
}

Eavesdrop_Approach_II(addr_t A, addr_t B)
{
    Eavesdrop_Approach_I(A,B);
    BASR::register_overhear_callback(TCP|UDP, A, B,
        disclose_data);
}

struct RREQ {
    addr_t src;      // the originator
    addr_t dst;      // the target
    int src_seq;
    int dst_seq;
    addr_t ip_src;  // the forwarder
};

Eavesdrop_Approach_III(addr_t A, addr_t B)
{
    Eavesdrop_Approach_I(A,B);
    addr_t M=BASR::read_local_entry()->dst;
    int aseq=BASR::read_route_entry(A)->seq;
    int bseq=BASR::read_route_entry(B)->seq;
    RREQ rreq(B, A, bseq+1, aseq+1, M);
    broadcast(rreq);
}
```

5. Attack Library

The Attack Library in S-MobiEmu is a well-organized and extensible collection of carefully designed attacks and test cases. It also provides the structure to assist researchers in developing new test cases in a new study.

The attack library organizes attacks and test cases in a hierarchy structure based on their composition. The core of the attack library is a collection called *Attack Foundation Library*, which contains all the *atomic* attacks that define the basic attack behavior on a single node. These attacks can be used as building blocks to construct *compound* attacks or complex test cases. These more sophisticated attacks can also span over multiple nodes.

5.1. Methodology

The attacks included in attack library are used as test cases to test the security aspect of a MANET system. It is therefore very important that we carefully design and choose the set of test cases. Here, we use a methodology called attack taxonomy.

General attack taxonomy can be used to design test cases. Although each application environment has a different threat model, it is important to study a general attack taxonomy that can serve as a starting point for analyzing application-specific attack scenarios. For this purpose, we adopt the attack taxnomy developed in [7]. It is based on the goals of the attackers, that is, what the attackers aim to accomplish. Based on the taxonomy, we can enumerate possible basic attacks. By basic attacks, we mean action blocks that cannot be divided further, for instance, the delivery of a data packet or a reply to route request. In contrast, other attacks are compound attacks that are composed of a number of basic attacks and even some atomic normal actions. Therefore, even if we cannot enumerate (or anticipate) all possible attacks, our claim is that they must be built based on one or a few basic attacks, and therefore it is possible to develop security countermeasures for unknown attacks as well.

We now explain how we build the attack taxonomy for ad-hoc routing [7]. First, routing is viewed as a *process* involving causally related operations from a number of nodes. Any routing process can be decomposed into a series of basic routing events. A *basic event* is defined as the smallest set of causally related routing operations on every node. Note that a basic event may involve delivering or receiving one or multiple network packets. A series of network operations is identified as a single normal basic event only when they are conducted in a transactional fashion specified by the protocol logic. Otherwise, they are considered an *anomalous basic event*.

Then, an anomalous basic event can be classified from two dimensions, its target and operation. The routing behavior of MANET typically involves three elements that are also the targets for adversaries: routing messages, data packets, and the routing table. The possible attack operations on these targets can be identified by examining the following well-known security goals: confidentiality, integrity, and availability.

Next, we choose basic attacks that correspond to each category of anomalous basic events. We take a realistic view to implement only those basic attacks that are (currently) meaningful.

- Attacks on confidentiality: can be performed on data packets, routing messages and routing table entries. The data compromise expects to disclose confidential data, which only the source and destination nodes should have been able to access. However, information in routing messages is, by its essence, publicly accessible since every node may potentially benefit from the information for further routing decisions. In the first glance, the lack of confidentiality in routing messages appears to be harmless. However, [2] pointed out that routing information can be used to disclose location information about other nodes, which is crucial in, for example, a military scenario. Similar argument also applies for the confidentiality of routing table entries.

- Attacks on integrity: can also be performed on all targets: data packets, routing messages, and routing table entries. In general, [7] enumerates possible compromise operations in this category: fabrication, modification and removal. Here *integrity* compromise has a more general meaning: not only the compromise of the data integrity, i.e., modification of fields in an existing element, but also of the integrity of normal program logic (which includes fabrication of new elements and removal of old elements). Furthermore, there are two meaningful variations of "modification" when we implement modification of routing messages: on its contents, and on its deliver frequency (The rushing attack described [5] is such an example). Rushing attacks are typically not meaningful on data packets and thus we do not list them separately.

- Attacks on availability. One of the most common and well-known availability compromises is flooding, which is implemented by sending huge amount of traffic of data packets and routing messages that exceeds the normal traffic processing capacity by targeted victims. The availability compromise can also be conducted on routing table entries by overflowing the table with useless routes, which may result in a new route being discarded.

Following the above taxonomy, we define a set of basic attacks. Each basic attack corresponds to a different category of anomalous basic events. We can further construct compound attacks that are composed of a number of basic attacks. Even if we cannot enumerate (or anticipate) all possible attacks, our belief is that most of them consist of one or a few basic attacks. Therefore, by studying these basic attacks, it is possible to develop security countermeasures for unknown attacks as well.

5.2. Attack Foundation Library for Ad-hoc Routing

We used the attack taxonomy to build an attack foundation library that includes the basic attacks. We take a realistic view to implement only those basic attacks that are (currently) meaningful. Figure 4 lists all basic attacks in the attack foundation library.

The parameters shown here are rather self-evident. For example, `src` and `dst` are the IP addresses for the source and destination hosts.

Basic attacks on network confidentiality are easy to implement because they only passively listen to the routing or data traffic – we can simply call BASR function `register_callback()` to listen to local routing control messages and `register_overhear_callback()` to eavesdrop in data and routing control messages on neighboring nodes. Similarly, an attack on the confidentiality of routing table entries can be achieved by calling `read_route_entry()`.

`Route_Drop_*` and `Data_Drop_*` attacks are also easy to implement. They register a callback function with `register_callback()` for outgoing messages with corresponding packet types. The callback function will return DROP under a certain probability, otherwise ACCEPT is returned. The kernel who receives the output keywords DROP and ACCEPT will take the corresponding actions, i.e., to drop or to deliver the packet.

`Modify_*` attacks modify routing information in outgoing routing messages. They also register a callback hook for the interested packets. The callback function will return MODIFY that will, in turn, change the routing packets with the requested fields. `Change_*` attacks, on the other hand, change information directly in local routing table entries. It first calls `read_route_entry()` to copy information that does not need to be modified from the current routing table entry, then calls `write_route_entry()` to change relevant fields.

For basic attacks that fabricate routing messages such as *Route Request* (RREQ) and *Route Reply* (RREP), parameters `src` and `dst` represent the originator and the target respectively, while parameters `ip_src` and `ip_dst` specify the source and destination in the IP header, i.e., the previous hop and the next hop in a route path. Here, it is possible to specify a different address from the host's own address (known as IP spoofing). These attacks are implemented using basic network socket operations. Essentially, a special UDP packet is created to mimic the same packet formats of RREQ and RREP delivered by the routing protocol. In particular, RREQ packets are delivered to a broadcast address. RREP packets are delivered to the specified `ip_dst`. `False_Reply` attempts to reply a RREQ even if it is the destination and it does not have an available route. It simply copies the corresponding fields from the RREQ except

1. Confidentiality Compromises
   - (a) Attacks on Routing Messages
     - i. Location Disclosure
   - (b) Attacks on Data Packets
     - i. Data Disclosure
   - (c) Attacks on Routing Table Entries
2. Integrity Compromises
   - (a) Attacks on Routing Messages
     - i. Fabrication of Routing Messages
       - A. **`False_Request(src, dst, src_seq, dst_seq, ip_src)`** – Forge a *Route Request* even if there is no need to discover a new route.
       - B. **`Active_Reply(src, dst, dst_seq, ip_src, ip_dst)`** – Forge a *Route Reply* even if there are no related incoming Route Request messages.
       - C. **`False_Reply(rreq, dst_seq)`** – Forge a *Route Reply* for a Route Request message even if the node is not supposed to reply.
     - ii. Interruption of Routing Messages
       - A. **`Route_Drop_R(percentage, type)`** – Drop a percentage of routing packets with a certain type randomly.
       - B. **`Route_Drop_S(percentage, src, type)`** – Drop a percentage of routing packets with a specific source address.
       - C. **`Route_Drop_D(percentage, dst, type)`** – Drop a percentage of routing packets with a specific destination address.
     - iii. Modification of Routing Messages
       - A. **`Modify_Sequence_R(type, dst, dst_seq)`** – Modify the destination's sequence number.
       - B. **`Modify_Sequence_M(type, dst)`** – Increase the destination's sequence number to the largest allowed number.
       - C. **`Modify_Hop(dst, hop)`** – Change the hop count to a smaller value.
     - iv. Rushing of Routing Messages
       - A. **`Rushing_F(dst)`** – Shorten the waiting time for *Route Replies* when a route is unavailable.
       - B. **`Rushing_Y(dst)`** – Shorten the waiting time to send a *Route Reply* after a *Route Request* is received.

2. Integrity Compromises (continued)
   - (b) Attacks on Data Packets
     - i. Fabrication of Data Packets
     - ii. Interruption of Data Packets
       - A. **`Data_Drop_R(percentage, type)`**, **`Data_Drop_S(percentage, src, type)`**, **`Data_Drop_D(percentage, dst, type)`** – Similar to Route_Drop attacks, but drop data packets instead.
     - iii. Modification of Data Packets
   - (c) Attacks on Routing Table Entries
     - i. Add Route
       - A. **`Add_Route_I(dst)`** – Randomly select and validate an invalid route entry.
       - B. **`Add_Route_N (new_rentry)`** – Add a route entry directly with random destination address.
       - C. **`Add_Route(dst, next_hop, dst_seq)`** – Either validate or add a route entry, depending on whether the routing entry has existed.
     - ii. Remove Route
       - A. **`Delete_Route(dst)`** – Invalidate a random valid route.
     - iii. Change Route Cost
       - A. **`Change_Sequence_R(dst, dseq)`**, **`Change_Sequence_M(dst)`**, **`Change_Hop(dst, hop)`** – Similar to Modify attacks, but the change is directly applied on the routing table entries.
3. Availability Compromises
   - (a) Attacks on Routing Messages
     - i. Routing Message Flooding
       - A. **`Route_Flooding_R(freq, type)`** – Flood with both source and destination addresses randomized.
       - B. **`Route_Flooding_S(freq, src, type)`** – Flood with the same source address and random destination addresses.
       - C. **`Route_Flooding_D(freq, dst, type)`** – Flood to a single destination with random source addresses.
   - (b) Attacks on Data Packets
     - i. Data Packet Flooding
       - A. **`Data_Flooding_R(freq, type)`**, **`Data_Flooding_S(freq, src, type)`**, **`Data_Flooding_D(freq, dst, type)`** – Similar to Route_Flooding attacks, but with data packets.
   - (c) Attacks on Routing Table Entries
     - i. Routing Table Overflow
       - A. **`Overflow_Table()`** – Add excessive routes to overflow the routing table.

Figure 4. Attack Foundation Library for Ad-hoc Routing

that the destination sequence number can be specified explicitly.

5.3. Extending the Attack Library

The attack library is extensible because compound attacks can be built from the basic attacks in the foundation library or other attacks. Here we will present several realistic attacks that we developed and included in the extended attack library. They can serve as the common test cases to test, evaluate and compare different security solutions in their response to MANET threats. Furthermore, they can also be used as building blocks of more complicated attacks.

We now show a few attack examples in pseudo codes.

Route Invasion: Inject a node in an active route.

```
Route_Invasion(double duration /*unused*/,
    addr_t src, addr_t dst)
{
    if  !read_route_entry(src) ||
        !read_route_entry(dst)
    {
        return NO_ATTACK;
    }
    cur=read_local_entry()->dst;
    cseq=read_local_entry()->seq;
    sseq=read_route_entry(src)->seq;
    dseq=read_route_entry(dst)->seq;
    False_Request(dst, src, dseq+1, sseq+1, cur);
    False_Request(cur, dst, cseq, dseq+1, cur);
}
```

If the route from `src` to `dst` exists, the attacker first generates a `False_Request` basic attack with a larger sequence number for `dst`. It will make all nodes, including `src`, update their routes to `dst` using `cur` as the next hop. Then, the attacker generates a second `False_Request` attack, which will launch a route discovery process to establish the route from `cur` to `dst`. Eventually, `cur` will be injected in the route from `src` to `dst`.

Note that this script does not prevent the route to be changed back later. We can implement a persistent version of this attack by calling the basic script repeatedly. The pseudo code looks like this:

```
Route_Invasion_P(double duration,
    addr_t src, addr_t dst)
{
    while(duration>0)
    {
        Route_Invasion(0, src, dst);
        sleep(period);
        duration=duration-period;
    }
}
```

Similar techniques may be applied to many other attacks as well.

Route Loop: Create a route loop.

```
Route_Loop(double duration /*unused*/,
    addr_t src, addr_t dst)
{
    if  !read_route_entry(src) ||
        !read_route_entry(dst)
    {
        return NO_ATTACK;
    }
    cur=read_local_entry()->dst;
    prev=read_route_entry(src).next_hop;
    next=read_route_entry(dst).next_hop;
    dseq=read_route_entry(dst).seq;
    Add_Route(dst, prev, dseq+1);
    Active_Reply(src, dst, dseq+1, cur, next);
}
```

If the attacker is close to a route from `src` to `dst` such that two subsequent nodes in this route, `prev` and `next`, are in the attacker's 1-hop neighborhood, the attacker can first add a route to `dst` using `prev` as the next hop. It then generates an `Active_Reply` basic attack to `next`, using a larger sequence number for `dst` in the RREP message. It will make `next` update its route to `dst` via `cur`. When `prev` receives a packet from `src`, the packet is forwarded according to the normal path and it will eventually reach `next`. However, `next` now thinks the best route to `dst` is through `cur` and `cur` forwards it back to `prev`. This effectively creates a loop from `src` to `dst` and all packets will be dropped in the route when their TTLs drop to zero.

A similar attack can be implemented when the attacker is not close to the targeted route. The attacker can first find a victim node V that is close to the route. Instead of calling `Add_Route` locally on V (which will require an additional compromise on V), the attacker can use either `False_Request` or `Active_Reply` to force V to update its route to `dst` via V's corresponding `prev`. The rest is similar.

Sinkhole: Create a sinkhole that redirects all neighboring traffic to a particular node.

```
Sinkhole(double duration /*unused*/, addr_t victim)
{
    cur=read_local_entry()->dst;
    sseq=read_route_entry(victim)->seq;
    dst=random address that does not exist;
    dseq=random sequence number;
    False_Request(victim, dst, sseq+1, dseq, cur);
    Data_Drop_D(1.0, victim, TCP|UDP);
}
```

The attacker generates a `False_Request` that appears to come from the `victim` and to a non-existent destination. Since nobody has a route to that destination, the RREQ will eventually flood throughout the whole network. As a side effect, all nodes that receive the RREQ will update its route to the victim via `cur`. Eventually, `cur` becomes a Sinkhole for `victim`.

Note that the above attack examples only act on a single host. However, it is not difficult to develop a powerful distributed attack with two or more compromised hosts based on similar techniques.

6. Measurement and Evaluation Tools

Performance measurement tools are designed to evaluate the effectiveness of the security solution in maintaining application mission objective when under attacks. They are very useful to compare alternative security solutions. Since different security solutions have different requirements and may target different ranges of attacks or threats, there is no single measurement that can be used alone to determine the best solution. To provide an objective basis for decision-making, we should support multiple measurement tools based on different performance models. How to prioritize and assess these metrics wisely and choose the best security solution(s) is a research problem that goes beyond our paper.

In S-MobiEmu, we build a set of measurement tools based on the cost-benefit analysis model [10]. They can be extended to build other measurement and evaluation tools.

Every security solution comes with a cost. We can identify the major cost factors as *response cost* and *operational cost* [10]. Response cost is the cost to perform responsive actions based on the intrusion evidence indicated by the security solution. Operational cost is the cost of applying security functions (e.g., encryption or intrusion analysis).

On the other hand, there is also a benefit by deploying a security solution. One benefit measurement is *damage cost*, which describes the degree of damage to the system that is caused by an attack when the security solution is not available. Another measurement is *effectiveness*, which describes how effective the security solution can reduce the damage cost of a particular attack.

In our framework, we consider only the *objective* measures that are relevant to routing security. In particular, we do not include the *response cost* and *damage cost* because they are application and environment specific, and can thus be *subjective*.

Most metrics in the tree are self-evident. We describe the operational cost in the amount of resource consumption, which can roughly be classified as system resource (such as CPU, memory, disk, etc.) consumption, and network resource (such as incoming and outgoing network traffic) consumption. In particular, we consider the the amount of overhearing traffic as an overhead as well. The usefulness of this metric can be shown by the energy efficiency problem. In wireless networks, energy efficiency is a very important issue. It is widely agreed that both communication overhead and overhearing overhead contribute to the majority of energy consumption in a MANET environment. Therefore, energy consumption can be measured (approximately) in terms of both communication overhead and overhearing overhead.

1. Operational Cost
   (a) System Resource Consumption
      i. CPU Usage
      ii. Memory Usage
   (b) Network Resource Consumption
      i. Communication Overhead
      ii. Overhearing Overhead
2. Effectiveness
   (a) Detection Accuracy
      i. Detection Rate
      ii. False Alarm Rate

Figure 5. Performance Measurement and Evaluation Library for Ad-hoc Routing

7. Case Study: Using S-MobiEmu to Evaluate Intrusion Detection Systems (IDS)

7.1. Overview of IDS Research Objective

The MANET environment is known to be more vulnerable than traditional wired networks, due to its dynamic and distributed nature [16, 14]. Many recent research efforts (such as [3, 4]) attempted to apply cryptographic solutions to secure MANET, especially on routing protocols. However, such intrusion prevention methods are usually just the first line of defense – this alone is often not sufficient. As systems become ever more complex, and as security is still often the after-thought, there are always exploitable weaknesses in the systems due to design and programming errors, or various “socially engineered” penetration techniques.

Intrusion detection can be used as a second wall to protect network systems because once an intrusion is detected, response can be put into place to minimize damages. The primary assumptions of intrusion detection are: user and program activities are observable, for example via system auditing mechanisms; and more importantly, normal and intrusion activities have distinct behavior. Intrusion detection therefore involves capturing audit data and reasoning about the evidence in the data to determine whether the system is under attack.

Designing an intrusion detection system (IDS) in MANET is a challenging task [14]. First, unlike wired network, MANET does not have traffic concentration points where the IDS can collect audit data for the entire network. Therefore, at any one time, the only available audit trace will be limited to communication activities taking place within the radio range, and the intrusion detection algorithms must be made to work on this partial and localized information. Further, IDS requires a well-defined attack taxonomy. In a new environment such as MANET, traditional attack analysis is not effective because it relies heavily on details of known vulnerabilities and attack incidents. Although MANET has many potential applications, none of them are widely used yet. As a result, only limited MANET attacks have been studied in the literature.

The objective of our IDS research is to investigate IDS techniques and to develop IDS-based security systems for MANET. Over the past several years, we have developed several such techniques. In particular, we have proposed two IDS frameworks for MANET [7, 6]. The first is a *node-based* framework where IDS agents are deployed on every node and they only utilize local information [7]. This framework further contains a specification-based approach that can accurately detect attacks that are direct violations of the protocol specification, and a statistics-based approach that can use machine learning tools to detect attacks that are temporal and statistical in nature. The second IDS framework is *cluster-based* because there are certain attacks whose patterns can only be detected through collaborative efforts among multiple nodes [6]. In this framework, IDS agents can collect and use features from multiple nodes in a neighborhood.

In this case study, we are to evaluate these two frameworks in real systems. We have implemented the two frameworks in the S-MobiEmu platform, and have conducted S-MobiEmu experiments to evaluate their performance.

7.2. Experiment Setup

The IDS implementation relies heavily on BASR. In particular, the IDS agent is implemented as a separate process from the routing protocol (AODV), while BASR provides the necessary interface to learn the necessary information about the routing protocol by an external process.

The following parameters are used throughout our experiments. Mobility scenarios, unless specified explicitly, are generated using a random waypoint model with 50 nodes moving in an area of 1000m by 1000m. The pause time between movements is 10 seconds and the maximum movement speed is 20.0 m/s. Randomized TCP and UDP/CBR (Constant Bit Rate) traffic are used. We create 20 connections and the average traffic rate is 4 packets per second. These parameters define a typical MANET scenario with modest traffic load and mobility, which are similar to those used in other experiments [13].

Five training data sets, each of which runs 10,000 seconds, are generated for training statistical detection models. Each data set contains randomly generated attack instances with random start time and random duration periods. The number of anomalous records accounts for roughly 50% of total records. We also use ten attack data sets and two normal data sets as the test data. Each test set runs 5,000 seconds. Normal data sets do not contain any attacks. In each attack data set, different types of attacks are generated randomly with equal probability and attack instances are generated with random duration periods. Compared with the training data, a larger proportion, i.e., 80% of total records are normal in an attack test set. It reflects a more realistic setting since normal events should be the majority in any real network environment.

7.3. Research Results

In Section 7.1, we outlined two IDS frameworks, node-based and cluster-based. Intuitively, the cluster-based framework is more powerful but potentially less resource efficient. In order to evaluate these solutions, we need to evaluate different performance measurements. For simplicity, we present the results on two performance categories, *detection accuracy* and *network resource consumption*.

Detection Accuracy: We are able to detect a number of basic and compound attacks using the specification-based approach. They include `Data_Drop_*`, `Route_Drop_*`, `Add_Route_*`, `Delete_Route`, `Overflow_Table`, `Change_Sequence_*`, `Change_Hop`, `False_Request`, `Active_Reply`, `False_Reply`, `Route_Invasion`, `Route_Loop`, and `Sinkhole`. We manually verified that our specification is accurate in terms of representing all allowable normal operations. Therefore, the *detection rates* for these direct violations of protocol specification are 100% and the *false alarm rates* are 0%.

The statistics-based approach further detects the following attacks: `Data_Flooding_*`, `Route_Flooding_*`, `Modify_Sequence_M`, and `Rushing_*`. By controlling the *false alarm rate* to be less than 1%, these attacks can be detected with an average *detection rate* of 91%.

Note that these attacks can only be detected on the node it is launched. A possible improvement can be achieved with the cluster-based framework. It can detect attacks not only on the compromised node, but also on other nodes through collaborative efforts. For example, the cluster-based IDS detects the Sinkhole attack with 91% *detection rate* and less than 1% *false alarm rate*. Although the accuracy is slightly lower than that of the node-based approach, it can prevent a single point of failure.

Network Resource Consumption: Although the node-based IDS does not deliver or receive any extra network messages, it requires the use of promiscuous mode, which also consumes significant amount of energy due to overhearing overhead. In contrast, although the cluster-based IDS requires additional communication between IDS agents, only a subset of nodes are required to enable the promiscuous mode. We compare the overall network overhead as a rough estimation of energy consumption by including both *communication overhead* and *overhearing overhead* between node-based and cluster-based frameworks in Figure 6, where *m* is the maximum number of clusterheads per cluster. The figure shows that a cluster-based approach can even be more energy efficient than a node-based approach under modest mobility levels.

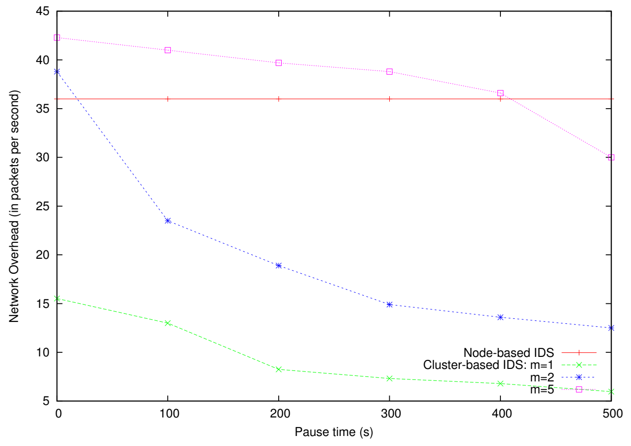

Figure 6. Overall Network Overhead

7.4. Experience of Using S-MobiEmu

We expect our IDS can detect routing anomalies by utilizing information on both the internal states of the underlying routing protocol and the patterns of network events. In our implementation, we found that the BASR approach serves us very well for this purpose. We can fully reconstruct the protocol specification indirectly through BASR hooks and use the specification to detect anomalies. The implementation is non-trivial but it can be done fairly efficiently.

We also experimented with a similar intrusion detection system on the simulation platform ns-2. Compared with that experience, development using S-MobiEmu is easier, because of fewer resource constraints. By using the *user-mode Linux extension* to MobiEmu [15], we were able to experiment on an emulation platform of as many as 100 virtual nodes. Since each test experiment can be conducted in real-time, it turns out to be much faster than a simulated run. Thus, we were able to conduct a larger number of experiments with a wider parameter selection.

We further state that our implementation with BASR has additional security advantages than a straightforward implementation without BASR. We note that a traditional IDS solution requires a trace log from the routing protocol process as input. Let us assume an attacker may not have the source code to the routing protocol and therefore cannot tamper with the normal protocol behavior directly. However, the attacker may still be able to obtain the needed privileges to modify the trace log file right before IDS can access it. This attack will not succeed in our implementation because we do not use the trace log as an intermediate audit log file. Instead, the IDS uses (read-only) helper hooks directly from the routing protocol.

8. Discussions

8.1. Code Complexity

The BASR module for AODV is about 400 lines in C. The attack library, which includes the implementation of 28 basic attacks, which form the *Attack Foundation Library* and about 10 compound attacks, is implemented in about 3,500 lines in C++. The performance measurement toolkit contains about 800 lines of code. For our case study, the node-based IDS has about 15,000 lines of code. The cluster-based IDS has about 8,000 lines of code, excluding the shared code base from the node-based IDS.

The source code of S-MobiEmu will be available at MobiEmu website `http://mobiemu.sourceforge.net`. We also plan to release our IDS software in the future.

8.2. Limitation

We would like to point out that S-MobiEmu is not suitable for studying attacks in physical layer (such as jamming), because the wireless communication is emulated. However, if we replace the network emulator (MobiEmu) with a real deployed MANET network, it is possible to use the rest of S-MobiEmu platform to run experiments, but such experiments may not be reproducible for the reasons we have explained earlier. Similarly, we may have to run real experiments for MAC-layer security study, because today's wireless MAC is almost always implemented in firmware and is inaccessible. However, if the MAC protocols are implemented in host OS, like in some new architecture such as "Native WiFi", we may be able to use S-MobiEmu in emulation mode. We also envision that S-MobiEmu can be extended to support MAC-layer security study in future software-defined radio platforms where MAC protocols are programmable in DSP or FPGA.

9. Related Work

To the best of our knowledge, there is no similar secure MANET testing system reported in the open literature. In wired network security, the best known test environment is perhaps LARIAT, an IDS testbed used in the 1998 and 1999 DARPA Intrusion Detection evaluation [11, 12]. LARIAT provides a configurable test environment where intrusion detection modules can be "plugged" in the testbed to capture audit data and invoke response. It provides many ways to configure background traffic and attack generation. However, it does not provide APIs to extend its attack library to accommodate more/new attacks.

10. Conclusion

In this paper, we have explained the needs to have an experimental environment to assist the development and evaluation of secure MANET. We have developed one such platform called S-MobiEmu. It allows us to test actual secure MANET code in repeatable experiments. It provides the necessary programming abstraction for us to design and implement attack test cases and the flexibility for us to extend the attack library in the future. We have tested S-MobiEmu in our own secure MANET research. We used it to evaluate an Intrusion Detection System and gained very positive results. We believe that S-MobiEmu will be a very useful tool for secure MANET research community.

Our future work will include a better programming abstraction for attacks. We will exploit a scripting language approach, where attack scenarios can be expressed in a form that is easily understandable by human and executable by machine. Then, the core of S-MobiEmu will be an attack engine that interprets the scripts and injects malicious traffic, security-related events, and the effect of security compromise (such as worm spread) into the emulated network. Further future work will also include a better performance measurement model.

Acknowledgment: This work is supported in part by NSF grants CCR-0133629 and CCR-0311024 and Army Research Office contract DAAD19-01-1-0610. The contents of this work are solely the responsibility of the authors and do not necessarily represent the official views of NSF and the U.S. Army.

References

[1] L. Breslau, et.al. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[2] A. Fasbender, D. Kesdogan, and O. Kubitz. Variable and scalable security: Protection of location information in mobile IP. In *Proceedings of the 46th IEEE Vehicular Technology Society Conference*, Atlanta, GA, Mar. 1996.

[3] M. Guerrero Zapata. Secure Ad hoc On-Demand Distance Vector Routing. *ACM Mobile Computing and Communications Review (MC2R)*, 6(3):106–107, July 2002.

[4] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom'02)*, Sept. 2002.

[5] Y.-C. Hu, A. Perrig, and D. B. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *Proceedings of the ACM Workshop on Wireless Security (WiSe'03)*, San Diego, CA, Sept. 19, 2003.

[6] Y. Huang and W. Lee. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'03)*, Oct. 2003.

[7] Y. Huang and W. Lee. Attack analysis and detection for ad hoc routing protocols. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, French Riviera, France, 2004.

[8] V. Jacobson, C. Leres, and S. McCanne. `tcpdump`. available via anonumous ftp to ftp.ee.lbl.gov, June 1989.

[9] V. Kawadia, Y. Zhang, and B. Gupta. System services for adhoc routing: Architecture, implementation and experiences. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys'03)*, San Francisco, CA, May 2003.

[10] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1,2), 2002.

[11] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX I)*, volume 2, Jan. 2000.

[12] R. P. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. J. Das. Analysis and results of the 1999 DARPA offline intrusion detection evaluation. In *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID'00)*, Oct. 2000.

[13] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (Mobicom'00)*, pages 255–265, 2000.

[14] Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (Mobicom'00)*, pages 275–283, 2000.

[15] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, Lausanne, Switzerland, June 2002.

[16] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.