

Securing Routing in Open Networks Using Secure Traceroute

Gaurav Mathur
BITS Pilani

Venkata N. Padmanabhan
Microsoft Research

Daniel R. Simon
Microsoft Research

Abstract—We consider the threat imposed on network routing in “open” networks such as community wireless networks. The key characteristic of such networks is that it is relatively easy for users (and attackers) to add routers, establish (possibly wireless) links, and advertise routes. We argue that the traditional focus on securing the routing protocol is insufficient to address the threats arising in this environment. It is also important to secure packet forwarding. To this end, we apply a *secure traceroute* protocol to detect and localize faulty packet forwarding, which can aid problem resolution either via automatic rerouting or via human action. We present a security analysis of the protocol, discuss our implementation of it in a community wireless network testbed, and show that secure traceroute imposes a negligible overhead on performance.

I. INTRODUCTION

Recent years have seen an increasing deployment and prevalence of what might be termed “open” networks. The key characteristic of such networks is that it is relatively easy for users to add routers, establish (possibly wireless) links, and advertise routes. This situation is quite different from that in much of today’s Internet, where ISPs and IT departments tightly control much of the routing infrastructure and users only control the endpoints. Examples of such open networks include wireless ad-hoc networks, community networks [26], [29] based on wireless (e.g., 802.11) and/or wireline (e.g., power line [28] or phone line [27]) link technologies, and peer-to-peer overlay networks [1].

Since users control the routers in open networks, it is relatively easy for attackers to disrupt routing by taking over control of existing routers or by introducing new (faulty) routers into the network. An attacker could do a number of things:

- 1) Disrupt the routing protocol (i.e., the *control plane* of the network) by introducing bogus routing messages or tampering with routing messages originating at other nodes. For instance, the attacker could advertise routes to destinations that it, or the node it is masquerading as, does not have a way of reaching, thereby creating a “blackhole”. Conversely it could suppress or modify routing messages to make any routes passing through itself seem less attractive, thereby lightening the load of traffic it needs to handle and becoming a “free rider”.
- 2) Disrupt packet forwarding (i.e., the *data plane* of the network) by dropping packets routed to it by

its neighbors. This dropping can either happen indiscriminately or selectively based on information such as the source or destination address.

Much of the existing work on routing security, both in the wired Internet and in wireless ad-hoc networks, has focussed on securing the routing protocol (e.g., [15], [9]). The idea is to ensure the authenticity of routing messages—say using digital signatures. However, authentication of the routing protocol messages is not sufficient to prevent the disruption of routing. Authentication of a routing protocol message does not guarantee its correctness. An attacker could steal the credentials of a legitimate user or a legitimate user could himself/herself turn malicious, and thereby inject authenticated but incorrect routing information into the network. Furthermore, an attacker could refrain from corrupting routing messages but still fail to forward data packets. Thus, beyond ensuring the security of the routing protocol, it is also important to deal directly with packet forwarding misbehavior. In particular, we need a way to securely detect and localize the source of packet forwarding misbehavior so that the problem can then be corrected by routing around the trouble spot, invalidating the (presumably compromised) credentials used to advertise the route through the trouble spot, or taking offline action at the human level.

Conceptually, a tool such as `traceroute` [12] could be used to detect forwarding misbehavior and identify the offending router. However, an attacker can provide a misleading impression of the problem by treating `traceroute` packets differently from normal packets or by tampering with the `traceroute` responses sent by other nodes. To avoid this problem, we use a *secure traceroute* protocol (“SecTrace” for short) that securely traces the path of *existing* traffic by having intermediate routers *prove* that they have received the traffic rather than depending on implicit responses. The operation of SecTrace is inconspicuous to all nodes except the one being traced at any point in time. SecTrace responses are also authenticated, to verify their origin and prevent spoofing or tampering.

We use SecTrace in the context of a community wireless “mesh” network. SecTrace is used in two ways. First, it is used by nodes to routinely monitor end-to-end connectivity to other mesh nodes that they are actively communicating with. Second, when a connectivity problem is detected, SecTrace is applied hop-by-hop to identify the offending router.

We have implemented SecTrace in a wireless mesh testbed consisting of Windows XP-based laptops equipped with 802.11 a/b/g radios. Our experimental results show that in practice SecTrace imposes a negligible overhead on the network's performance. We also describe a simple extension to the DSR-like [13] source routing algorithm used in our testbed to route around trouble spots identified by SecTrace.

The remainder of this paper is organized as follows. In Section II, we discuss the constraints that need to be placed on open networks, in particular community wireless mesh networks, to enable us to secure network routing. We discuss the design and operation of our SecTrace-based proposal for securing routing in such networks in Section III, and present a security analysis of the protocol in Section IV. We briefly discuss SecTrace authentication issues in Section V, followed by a description of our implementation of SecTrace in a wireless mesh network testbed and an evaluation of its impact on performance in Section VI. In Section VII, we present a discussion of attacks on SecTrace and how we might defend against the attacks. We survey related work on network routing security in Section VIII, and finally conclude with a summary of the paper in Section IX.

II. ASSUMPTIONS AND THREAT MODEL

An "open network" can be open to various degrees. For example, nodes may be able to join and leave the network at will, completely anonymously, with no central organization whatsoever. At the other extreme, a central membership registry (such as a Kerberos KDC) may reserve the right to grant and deny nodes the privilege of authenticating to, and thus participating in, the network, in well-defined roles. The more open the network, the more vulnerable it is to certain types of attack. For example, if nodes are free to create new identities for themselves at will, then any attempt to detect and "blacklist" a misbehaving node can be countered by a change of identity (e.g., the Sybil attack [7]).

For this reason, we assume that the nodes in the network are associated with some kind of revocable, verifiable, persistent, controlled-issuance identity. That is, honest nodes can authenticate each other's identities, and dishonest ones, if identified, can have their identities revoked, and cannot issue themselves new ones. A natural example of such an identity system is a public key infrastructure (PKI), although other frameworks, such as Kerberos, also fit the assumption.

While having routers certified by a PKI does restrict the "openness" of the network, such certification may nevertheless be prudent given the ease with which attackers could otherwise disrupt the network. The certification procedure can be low in management overhead and quite open. For example, a backbone of (closed) wireless mesh routers (see Section VI-A) can be certified as authentic by the vendor, containing a certificate verifying its unique ID (akin to the unique MAC address on Ethernet and 802.11 hardware).

These routers could then be automatically admitted into a community wireless network without requiring further user action. Alternatively, users in the community wireless network could set up their own procedure for certification.

We also assume that the network uses a single-path routing protocol of some kind. Networks where, for example, all traffic is propagated by flooding can achieve robustness (albeit at a potentially significant performance cost) in the complete absence of identities and quite possibly in the presence of numerous malicious adversaries. But single-path routing protocols have more difficulty dealing with individual misbehaving routers, since it is easier for the adversary to disrupt the forwarding of a stream of unrepliated packets along a common path. A mechanism to detect such misbehavior is therefore desirable.

Finally, we assume that the number of misbehaving nodes is relatively small. For example, we do not claim to be able to deal with, say, a network made up of one-third adversarial nodes, as in the literature on Byzantine fault tolerance. As we will see in Section III-A, we are assuming that enough nodes are honest that general connectivity in the network is maintained, so that the messages in our protocol can get through. If there are malicious nodes on most or all of the paths to a destination, it may be impractical or even impossible to eliminate the malicious nodes and find a working path to the destination. Our goal is to deal with fairly limited, localized routing problems in this setting; if the routing disruption is too widespread, involving too many nodes, then it can interfere with any attempt to investigate it.

The misbehaving nodes, however, may be arbitrarily malicious adversaries. They may use their links to other nodes to send arbitrary data, in arbitrary formats, up to the links' capacity. They may, for example, falsely attribute other nodes' identities to themselves, or originate traffic and claim to be forwarding it from another node. (Note, though, that the assumed identity infrastructure, if used properly, can enable other nodes to recognize such "spoofing" and reject it. Also, nodes whose identities are compromised and used by malicious nodes are considered to be themselves "dishonest", insofar as their identities are being used maliciously.)

A typical example of our target problem might be a single faulty router somewhere in the network (although our approach certainly does not assume that there is only one faulty router). Note that it is not necessary to distinguish between a merely faulty (e.g., misconfigured) router and a malicious one; in practice, after all, sufficiently erroneous behavior can be indistinguishable from malice. (An adversary may, for instance, attempt to disguise its malicious behavior as mere error in order to avoid later retribution.) Hence we assume faulty routers to be capable of arbitrary Byzantine behavior.

III. PROPOSED SOLUTION

A. Secure Traceroute

The normal traceroute protocol [12] involves the sender simply sending packets with increasing TTL (time to live) values, and waiting for an ICMP time-exceeded response from the router that receives the packet when the TTL expires. Normally, this protocol generates a sequence of addresses of nodes on the path to the destination, or at least up to the point where packets are being lost on a faulty link. However, a malicious router could intercept and alter traceroute traffic to give an arbitrary misleading impression—say, by letting only traceroute packets through, or by constructing fake responses to them so as to give the impression that they are getting through and demonstrating a fully functioning path (or a path with a problem elsewhere). Secure traceroute (SecTrace) is intended to prevent this type of disruption of the traceroute process by verifying the origin of responses, and preventing traceroute packets from being handled differently from ordinary traffic.

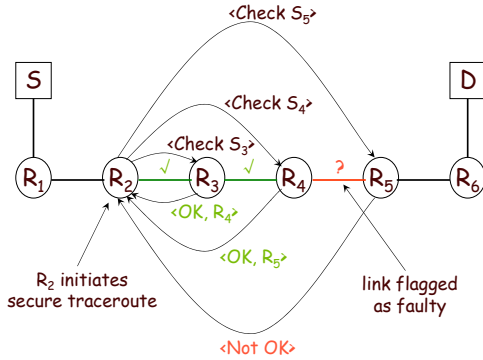


Fig. 1. An illustration of SecTrace in operation. R_2 initiates SecTrace and concludes eventually that the link between R_4 and R_5 is faulty. S_i denotes the “signature” of the packets that router R_i is asked to treat as traceroute packets.

The operation of SecTrace is illustrated in Figure 1. As in normal traceroute, SecTrace proceeds hop by hop, with each node on the path being asked to respond to traceroute traffic. However, there are several fundamental differences between them, including traceroute packet contents and the responses they elicit from routers. We outline the specifics of SecTrace below:

- 1) In addition to their own identity (which is included implicitly in the response packet), nodes responding to SecTrace packets provide a next-hop router identity for the packet. Thus the node performing the traceroute always knows the identity of the (expected) next router on the path. If source routing is used (as it is in our wireless mesh testbed), then the initiating node already has the entire route and hence the identity of the next-hop to be verified, as discussed in Section VI-C.
- 2) Prior to sending the traceroute packets, the tracing node establishes a shared key for encrypted, authenticated communication to and from the expected next

node. (How this is done in our testbed implementation is described in Section VI.) Using this key, identifying information, which specifies the “signature” of the packets to be treated as SecTrace packets, is securely passed from the tracing node to the expected next node. This information consists of the values (or constraints on the values) of certain fields on the packets. For example, it could be specified that all packets between certain source and destination addresses for which a certain field contains a particular value modulo a specific prime number should be examined. Alternatively, a random value (“nonce”) could be inserted into some field in the packet by the tracing node, and certain such values would be designated as signifying a SecTrace packet. Thus, SecTrace packets will look indistinguishable from ordinary traffic to intermediate nodes, and will therefore constitute a representative sample of traffic between the specified source and destination ranges.

- 3) In replying to a packet identified as a SecTrace packet, a node sends some agreed-upon identifying marker for the packet back to the tracing node, to “prove” that the packet has been received. Alternatively, some accumulated value based on multiple received packets may be returned. For example, if the tracing node is able to insert additional information in the SecTrace packets, then it can use a threshold secret-sharing scheme [22] to send a secret that can only be extracted when some minimum number of shares (and thus packets) has been received. (In a threshold secret-sharing scheme, a secret is “divided up” into a number of shares such that any subset of the shares of a certain “threshold” size suffices to reconstruct the secret, but any subset smaller than the threshold reveals no information about the secret. Threshold secret-sharing is an efficient operation, comparable in cost to symmetric-key cryptography.) By inserting one share per traceroute packet, and measuring how many shares must be sent before a reply is received, the tracing node can estimate what fraction of SecTrace packets are getting through.
- 4) In addition to packet-identifying information, the SecTrace reply contains a strongly secure Message Authentication Code (MAC) that ensures its authentic origin. The MAC is based on the original key shared between the tracing node and expected next node, and covers the entire response, including the address of the node to which the expected next node forwarded the traceroute packet, based on its destination. This new node becomes the “new” expected next node for the next step of the traceroute.

This iterative process produces, as does a normal traceroute, one of two possible results: either a complete route is determined, or a faulty link is found, such that one end of the link claims to be sending packets through the link, but

the other end claims not to be receiving them (or simply doesn't respond to the traceroute); we cannot be sure which end of the link is faulty. However, the identification of this link is much more reliable than for ordinary traceroute, since return packets are established to be coming from the correct node, and the use of normal packets in place of identifiable traceroute packets ensures that the route (whether functioning or faulty) accurately represents the route taken by normal packets.

IV. SECURITY ANALYSIS

To analyze the SecTrace protocol more formally, we must first describe a formal model for the network in which it is deployed. Our model consists of a collection of nodes, some possibly malicious and even colluding, and each with a unique name, or "address". The nodes are vertices in a directed graph; packets may be sent along the edges of the graph (in the correct direction) from node to node. The packets are of a fixed, limited size, and contain the address of a sender and a receiver. A routing protocol (which we will leave unspecified here) determines, for each honest node, the address of the neighboring node to which to pass on a packet received from another neighboring node, based only on information contained in the packet.

We consider a single step of the previously described informal protocol, in which a tracing node T attempts to determine if another node, P , is in fact on the path followed by a certain set of packets passing first through T . (We assume that P can distinguish between packets that have passed through T , assuming T is honest, and those that have not, and ignore, for the rest of this analysis, those that have not.)

Obviously, there is no way of proving that the packets are *not* reaching P ; P could, for instance, refuse to answer any queries from T . However, the protocol we describe allows P to prove (with a very low probability of error) that it is, in fact, receiving the packets in question.

We assume that P and T have authenticated each other, and share a secret key k . Also, we make no distinction between P and any other nodes with which it may be colluding, sharing either key or traffic information (or even inadvertently leaking such information); collectively, all such nodes, including all nodes that have (one way or another) obtained k , are together considered " P ". Finally, to simplify the cryptographic argument, we assume a random oracle Q , as in [2]. In practice, of course, the oracle is simulated by a cryptographic hash function, such as SHA-1.

The simplified protocol, then, is as follows: T sends P an l -bit string s , and P returns $Q(c_p)$, where c_p is the concatenation of k and p , for each packet p such that the first l bits of $Q(p)$ are identical to s . T verifies that P 's response is correct.

Theorem 1 *If P does not have access to a packet p , then the probability that P responds correctly in the above*

protocol is 2^{-m} each time, where m is the length of the output of Q .

Proof: (sketch) The proof is similar to the proof of "plaintext-aware" encryption schemes (as in [3]). Intuitively, if the "correct" input string (including both the packet p in question and k) was never used as input into Q , then the probability of P producing the correct output string is as stated. If some node (possibly P) did, in fact, query Q with the necessary input string, then that node had access to the packet in question *and* k , making it, effectively, a confederate of P . ■

V. AUTHENTICATING SECURE TRACEROUTE

The above protocol assumes that a secure (that is, encrypted, authenticated) key exchange can be performed between the tracing host and the expected next host. Ideally, a public-key infrastructure (PKI) would be available to allow such key exchange to occur using standard protocols (e.g., IPSEC [14]). Such a PKI for infrastructure operators (as opposed to end users) is not unthinkable, as the widely deployed "SSL" PKI for Web servers demonstrates. As we briefly described in section II, for example, one approach to PKI establishment in the mesh network setting is to equip "mesh routers" with asymmetric key pairs and public-key certificates at manufacture time, so that they can recognize each other as supporting various mesh protocols (such as secure traceroute).

In the absence of a PKI, various ad hoc mechanisms can be used. For example, PGP-style "Web of trust" techniques [31], [30] can be used to propagate routers' public keys from node to (trusting) node; using Web- or email-based protocols, nodes can distribute public keys of nodes they have established confidence in the sources of, and receive such keys in turn from others whose judgments they trust. Similarly, certain widely trusted hosts could voluntarily act as "key servers", collecting and verifying public keys of nodes and offering them for authenticated download to nodes that trust the key server. Finally, the redundancy of the network can be used to attempt to determine the valid public keys of other hosts at authentication time. By requesting the public key of a host multiple times, via multiple paths—possibly with the help of a set of widely distributed, trusted routing intermediaries (an overlay network of sorts)—a tracing host can increase the likelihood that the public key being returned has not been tampered with en route by a malicious host. Once the expected next host's public key has been obtained, a (one-way) authenticated key exchange is easy, and the secure traceroute can proceed.

VI. IMPLEMENTATION

We have implemented the secure traceroute protocol in a wireless mesh setting. Section VI-A presents some background information on the mesh setting. The assumptions, simplifications, and domain-specific decisions are

described in Section VI-B, while the actual implementation is described in Section VI-C.

A. The Wireless Mesh Setting

The wireless mesh setting we consider is a prototype of a community wireless network. It consists of end hosts, mesh nodes (routers), and Internet gateways. Figure 2 illustrates the architecture of the mesh network. Note that the mesh architecture and its routing protocol are not a contribution of our work. We merely provide sufficient background here to explain our implementation of SecTrace.

The mesh nodes (i.e., routers) form the backbone of the wireless mesh network. In addition to routing traffic through the mesh, each mesh node also acts as an access point, allowing (off-the-shelf and mesh-agnostic) end hosts in people’s homes to send and receive traffic through the mesh. These end hosts do not perform any routing function for the mesh, so we do not discuss them much here.

The wireless mesh also has a special node called the Internet Tap, or ITAP, that connects the mesh to the Internet as well as to other meshes. Due to its critical role in the operation of the mesh, the ITAP is trusted by all nodes in its mesh.

Routing within the wireless mesh network is accomplished by the Mesh Connectivity Layer (MCL) [8], which operates between layers 2 and 3 of the network protocol stack. MCL does routing, traditionally a network layer (layer 3) function, while operating beneath and transparently to the network layer. At the same time, MCL operates above the link layer (layer 2) that is associated with each physical NIC (network interface card) on the node. MCL creates a virtual NIC abstraction and assigns a single *virtual address* to the mesh node, thereby hiding the details of the multiple underlying physical NICs from the network layer (e.g., IP). The MCL also does multi-hop routing through the wireless mesh transparently to the network layer. A key advantage of doing routing below the network layer is that we can run various network layers (IPv4, IPv6, etc.) without modification. All nodes in the wireless mesh are part of a single IP subnet and are a single IP hop from each other, regardless of the number of underlying wireless hops. So our implementation of SecTrace in this environment operates at the MCL level rather than the IP level, as elaborated in Section VI-B.1.

The routing protocol used by the MCL is a variant of Dynamic Source Routing (DSR) [13]. The key ideas here are to do source routing and to discover routes dynamically, i.e., on demand as packets need to be routed to a particular destination. To discover a route, a *route-request* message is broadcast throughout the network. The route taken by a route-request message is recorded in the message itself as it makes its way to the destination. The destination then returns the recorded route to the source in a route-reply message. Although nodes suppress duplicate route-request messages, the destination may receive multiple route-request messages routed through independent paths,

if any, and hence may return multiple *route-reply* messages to the source. The source is then free to choose the route to use, say based on a metric such as hop count. When originating a packet, the source specifies the chosen route as the source route.

To make it difficult for attackers to insert themselves into the wireless mesh, each mesh node is assigned a public key certificate by a trusted root. Neighboring mesh nodes establish a long-lived session key between themselves to authenticate individual packets using HMAC-SHA1 [17]. End-to-end communication between mesh nodes is also encrypted using AES [25], although end hosts could also use a mechanism such as IPSec for true end-to-end privacy.

While these security mechanisms help, they do not prevent attacks on the DSR-based routing protocol. We could use schemes such as Ariadne [10] to secure the routing protocol against such attacks. However, we do not discuss this further here, since securing the routing protocol alone is insufficient and our focus is on securing packet forwarding.

B. Secure Traceroute in the Wireless Mesh

We discuss the specific decisions we made for implementing SecTrace in our wireless mesh setting.

1) *IP path versus MCL path*: As explained in Section VI-A, the entire mesh network is a single IP subnet. The path between two end hosts connected to the mesh network consists of 3 IP-level hops: one hop from the source end host to the source mesh node (i.e., the mesh node to which it is directly connected), a second hop from the source mesh node to the destination mesh node, and finally a third hop from the destination mesh node to the destination end host. For example, in Figure 2 the end-to-end path between end hosts EH_1 and EH_3 consists of 3 IP hops: $EH_1 \rightarrow MN_1$, $MN_1 \rightarrow MN_3$, and $MN_3 \rightarrow EH_3$.

The middle IP hop typically consists of multiple wireless hops traversing intermediate mesh nodes. (In the above example, the MCL path corresponding to this IP hop is: $MN_1 \rightarrow MN_2 \rightarrow MN_3$.) A packet traversing the middle IP hop is encapsulated in an MCL frame, whose source and destination addresses are the virtual addresses (Section VI-A) of the source and destination mesh nodes. Our implementation of SecTrace operates at the MCL level and hence operates only on the middle IP hop. One consequence is that SecTrace does not run end-to-end. While we could have implemented a separate, end-to-end SecTrace at the IP level, we did not do so since the MCL-level SecTrace gives us almost the same diagnostic capability in the wireless mesh setting.

2) *Initiating SecTrace*: SecTrace can be initiated either by end hosts or by mesh nodes (i.e., routers). We consider each possibility in turn.

End hosts are in the best position to detect end-to-end connectivity problems. The detection can happen either at the application level or at the transport protocol (e.g., TCP) level. When a problem has been detected, the end host

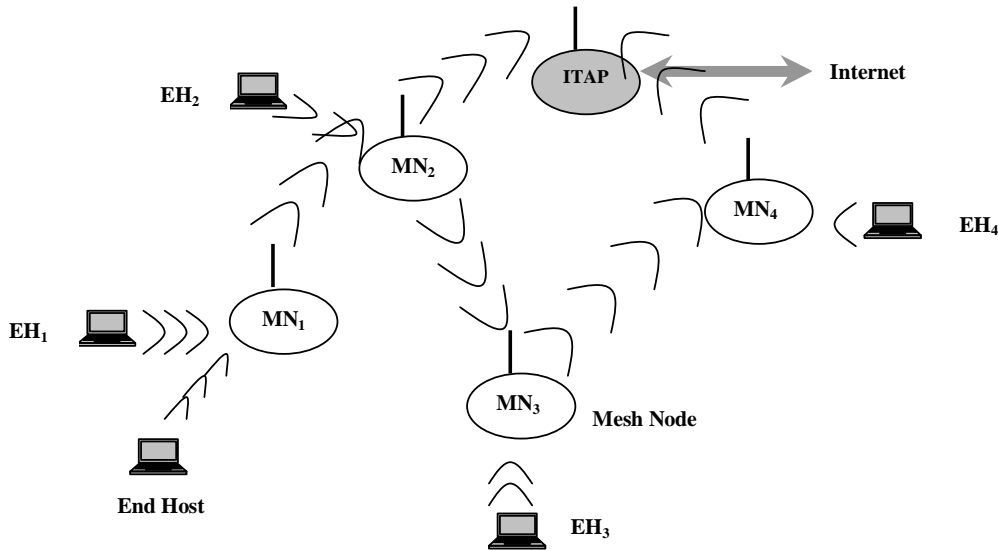


Fig. 2. Wireless Mesh Architecture

initiates SecTrace to the destination address to which it is experiencing connectivity problems. However, end-host-based detection and initiation faces the practical hurdle of having to instrument the network stack and/or applications on the end host.

To enable unmodified end hosts to benefit from SecTrace, we could have mesh nodes themselves detect connectivity problems and initiate SecTrace. However, it is difficult for mesh nodes to detect connectivity problems since they are not participants in the end-to-end communication. Even indirect inferencing, say based on observing the progression of sequence numbers on TCP data and ack packets, would be difficult if end hosts were to employ IPSec to encrypt traffic end-to-end. Our solution is to have mesh nodes use SecTrace to routinely monitor connectivity to other mesh nodes that they are sourcing data to. If this almost-end-to-end SecTrace reports a connectivity problem, then hop-by-hop SecTrace is used to locate the source of the problem.

Connectivity problems may often arise due to events such as congestion or link failure not induced by an attacker. Nevertheless, we believe that it is reasonable to use SecTrace as a general mechanism for localizing the problem since it provides greater assurance in its findings than non-secure techniques while still imposing a low overhead (see Section VI-D). Localizing the cause of congestion is useful since wireless mesh networks (and other open networks such as overlay networks) provide end hosts the opportunity to route around the congested link(s). This is in contrast to the Internet where routing control resides with the ISPs, making it difficult for end hosts to benefit from identifying the location of congestion, at least in the context of unicast communication with specific other hosts. We discuss the issue of routing around suspect links in Section VII-C.

3) Securing SecTrace Communication: Prior to any SecTrace communication, the public key certificates as-

signed to the mesh nodes are used to establish a bidirectional secure communication channel using the TLS 1.0 protocol [6] with two-way authentication.

While the TLS channel protects the privacy and integrity of communication between the investigating and investigated mesh nodes, it still provides attackers the opportunity to guess when SecTrace is in progress and thereby compromise its operation. Since much of the communication in the mesh network is likely to be between the mesh nodes and the ITAP (since users are likely to be interested primarily in Internet access), any direct communication between mesh nodes might be indicative of an impending SecTrace session, and thus be prone to targeted disruption by an adversary seeking to frame another node or being misled by an attacker who adjusts its behavior to avoid detection.

To make such attacks difficult, we make use of the trusted ITAP node as an intermediary. All SecTrace-related control communication between mesh nodes is routed via the ITAP. (Note that data packets, including the ones traced by SecTrace, are still routed directly between the mesh nodes and are *not* redirected via the ITAP.) Since all mesh nodes communicate frequently with the ITAP (to send and receive traffic to/from the Internet), it becomes very difficult for the attacker to detect the SecTrace packets. To make even traffic analysis difficult, we could pad the control packets to match the (predominant) size of the data packets (e.g., the network MTU), although we do not do so in our current implementation. This effectively prevents attackers from being able to frame an innocent router, thus avoiding the drawback of Perlman’s fault isolation scheme [20] discussed in Section VIII.

Note that the above procedure will work even when the ITAP runs SecTrace on paths *from* itself to other mesh nodes. However, the procedure breaks down when a mesh

node is unable to reach the ITAP and wishes to trace the path *to* it. We discuss more general solutions in Section VII-A, including “onion routing”-style encryption to prevent targeted framing of another node and initiating real and fake SecTrace attempts continually to make selectively adjusting behavior difficult.

4) Tracing SecTrace Packets: To enable the initiating router to designate a specific subset of packets as SecTrace packets, we need a field in the packet that changes randomly from one packet to the next but remains unchanged as a packet traverses multiple hops through the mesh. The MCL header of every wireless mesh packet contains such a field — an Initialization Vector (IV) field (Figure 3) used in the end-to-end AES encryption of the packet. A packet is designated as being a SecTrace packet if this field is equal to f modulo N , where f and N are negotiated between the investigating and investigated routers.

To enable the investigating node to verify that the investigated node has indeed received the designated packets, we considered using Shamir’s threshold secret sharing scheme [22]. The idea is for the investigating node to insert shares into the designated SecTrace packets and for the investigated router to compute and return the secret, which it can only do if it receives more than a threshold number of shares. A drawback of this scheme is that it requires an additional field containing the share to be included in *all* packets, whether designated as SecTrace packets or not and whether SecTrace is even active or not. Including this field selectively in packets, say to reduce the space overhead, might enable the detection of SecTrace packets via traffic analysis.

The alternative we chose to implement was to have the investigating and the investigated nodes first negotiate a secret key (16 bytes long). The investigated node then computes a keyed MAC (also 16 bytes long) on the payload of each SecTrace packet and returns these to the investigating node for verification. The specific MAC algorithm we use is HMAC-SHA1 [17], which is already available to us in the kernel since it is used by the MCL layer for authenticating packets on a hop-by-hop basis (Section VI-A). The main advantage of this MAC-based approach is that the byte-overhead of additional network messaging is only a function of the actual number of SecTrace packets. (For instance, with 50 SecTrace packets, the response returned by the investigated router would be a single, $50 \times 16 = 800$ byte packet.) No overhead is incurred for non-SecTrace packets whether SecTrace is currently active or not.

C. Implementation Details

We have implemented SecTrace on the Microsoft Windows XP OS platform. We discuss some specific details of the implementation here.

As discussed in Section VI-B.2, SecTrace can be initiated either by an end host or by a mesh node. Our implementation supports both these modes. An initiator program allows an end host to initiate SecTrace between the mesh node

that it is attached to and a destination mesh node.¹ Each mesh node runs a SecTrace daemon that can both initiate SecTrace to a target node and respond to SecTrace requests from other nodes.

The implementation of SecTrace on a mesh node has two components — a user-level Windows service (self-starting daemon) and an in-kernel device driver component (Figure 4). Communication between the two components happens via I/O control system calls (IOCTLs). The user-level service has a client sub-component that initiates SecTrace sessions with other nodes and a server sub-component that responds to SecTrace requests either from end hosts or from other mesh nodes. All SecTrace-related communication between mesh nodes happens only at the user level.

When the initiator contacts a mesh node to initiate SecTrace, TLS is used by the parties to authenticate each other. (We used Microsoft’s implementation of the TLS protocol on the Windows XP platform.) If this is successful, the mesh node then becomes the *investigating* router. It queries the MCL layer for the source route to the destination mesh node and pins this route to prevent route changes while SecTrace is in progress. It then establishes a SecTrace session, in turn, to each router along the path. Each of these routers is termed the *investigated* router while SecTrace to it is active. Figure 5 illustrates this process.

When the investigating router contacts an investigated router, TLS is used to establish a secure, two-way authenticated connection (routed via the ITAP, as explained in Section VI-B). The investigating router then sends to the investigated router (a) the source and destination addresses of the SecTrace packets (the source address being its own address and the destination address being that specified by the initiator), (b) the SecTrace filter parameters (the quantities f and N introduced in Section VI-B), and (c) a randomly generated key for use in the HMAC-SHA1 computation. At this point, the investigated router enables the SecTrace filter in its kernel to examine all *incoming* packets and to store the MACs (message authentication code hashes) of matching packets in an in-kernel buffer. After receiving a confirmation from the investigated router, the investigating router also enables the SecTrace filter to examine all *outgoing* packets and store the MACs of matching packets.

The user-level service on the investigating router periodically polls the kernel buffer that holds the MACs of the SecTrace packets. When the number of such packets exceeds a threshold (50 by default), it disables its in-kernel SecTrace filter. It then reestablishes a secure and two-way authenticated connection to the investigated router (again

¹In general, the initiator could contact any mesh node, not just the one to which it is attached. Such a capability would be useful, for instance, in enabling a network operator to trace various paths through the network from a central location. However, our discussion here is presented in terms of the more likely scenario of an end host contacting the mesh box to which it is attached in order to investigate a problem that it is experiencing and that no one else may have noticed.

Source Virtual Address	Destination Virtual Address	IV for AES
MAC (computed per hop)	Source Route to Destination	
... other fields of the MCL packet (2.5 layer) ...		
Layer 3 (IPv4, IPv6, etc.) Payload		

NOTE : The fields are not depicted according to actual proportions

Fig. 3. A Wireless Mesh Packet

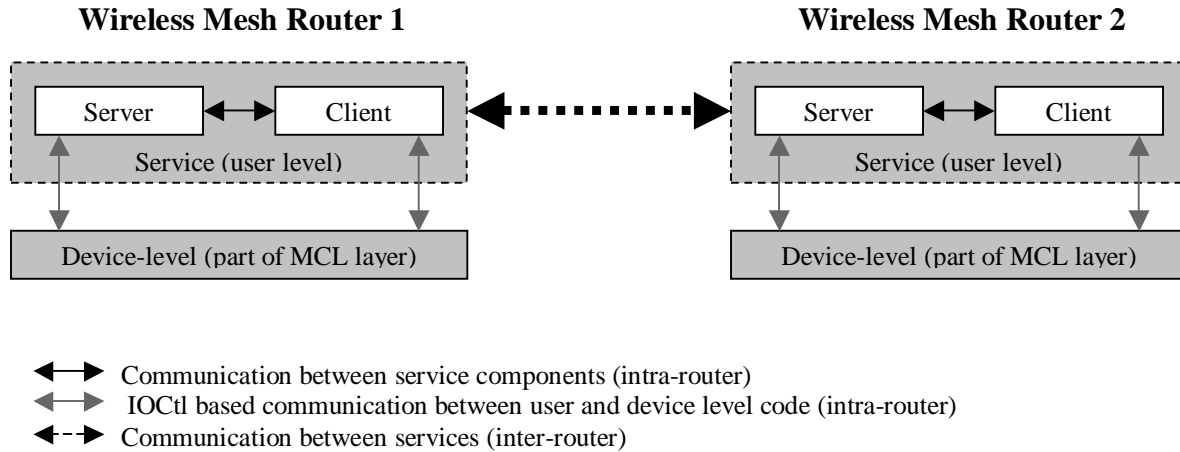


Fig. 4. SecTrace code architecture for the wireless mesh router

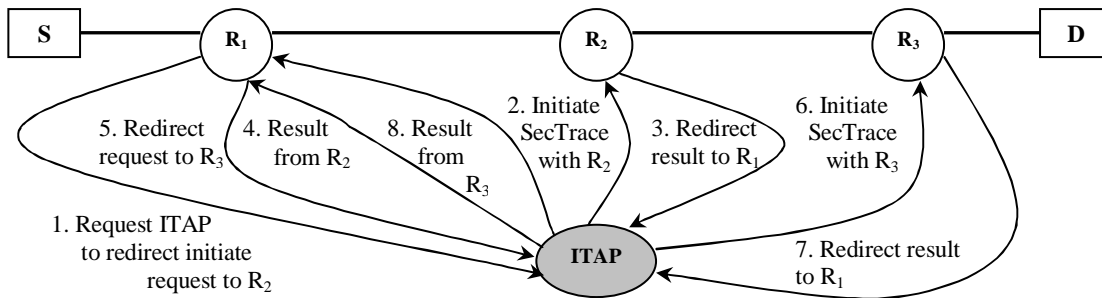


Fig. 5. An illustration of SecTrace in operation : S (the initiator) complains to R_1 (the investigating router) about connectivity to D . R_1 initiates a SecTrace to R_3 (the last node on the MCL path to D), asking the ITAP to redirect all SecTrace-related messaging. R_1 first verifies connectivity to R_2 and then to R_3 (the investigated routers).

routed via the ITAP). It asks the latter to also disable its SecTrace filter and then to return the MACs of all SecTrace packets it has received thus far. The investigating router checks to see what fraction of the MACs in its own list are returned by the investigated router. Since the investigating router enabled its filter after the investigated router and disabled it before the latter, any discrepancy in the list of MACs is likely to be due to actual packet loss rather than synchronization issues.

The investigating router repeats this procedure hop-by-

hop and eventually returns to the initiator statistics on the fraction of packets that reached each router along the path.

Our implementation also allows a node to participate in multiple simultaneous SecTrace sessions. The limit on the number of concurrent sessions is dependent on the amount of available memory in the kernel pool, and is set to 10 by default.

D. Performance Evaluation

To quantify the performance overhead of SecTrace, we measured the impact of the additional processing entailed

Fraction of packets traced	Throughput Overhead (standard deviation)
1/1	5.76% (0.95%)
1/2	3.51% (2.01%)
1/5	0.87% (1.93%)
1/10	0.78% (1.83%)

TABLE I

THE OVERHEAD IMPOSED BY SECTRACE, AS MEASURED BY ITS IMPACT ON THE THROUGHPUT OF A 16 MB TCP TRANSFER. WE CONDUCTED 10 RUNS OF THE EXPERIMENT FOR EACH CONFIGURATION. WE OBSERVE A NOTICEABLE IMPACT ONLY WHEN 50% OR MORE OF THE PACKETS ARE DEEMED TO BE SECTRACE PACKETS.

by SecTrace on TCP throughput. Note that our interest is in software-based routers (e.g., PC-based routers), not backbone-class high-speed routers (e.g., Cisco boxes) with a hardware-based fast path. For this experiment, we used two laptops with 2 GHz Pentium-4 processors connected via a 100 Mbps crossover Ethernet cable. We chose to run the experiment over Ethernet rather than over wireless to factor out the variability in throughput induced by the wireless medium. This variability is often large enough to make the overhead of SecTrace irrelevant. If the overhead imposed by SecTrace at Ethernet speeds turns out to be small, we can safely conclude that the overhead (in terms of impact on TCP throughput) would only be smaller at the lower wireless speeds.

We measured the throughput of a 16 MB TCP transfer between the two laptops, with and without SecTrace enabled the destination laptop. We conducted 10 runs of the experiment for each configuration. When SecTrace was enabled, the destination laptop would apply the SecTrace filter on each incoming packet and compute the HMAC-SHA1 hash on the matching packets (i.e., the “traced” packets), as discussed in Section VI-B. We varied the parameter N (Section VI-B) from 1 through 10, thus varying the fraction of traced packets from 100% to 10% on average. Table I shows the reduction in the throughput due to SecTrace compared to the case where SecTrace is disabled. We observe a small impact of 3-5% when 50-100% of the packets are traced. The overhead drops off quickly and becomes negligible (relative to the variability between multiple runs of the experiment) as the fraction of packets traced becomes smaller (as we would expect it to be in practice). Note that even when a multi-hop path is being traced, SecTrace is active (and thus incurring overhead) only on single node at any point in time, so the overhead does *not* add up over multiple hops.

Note that the above measurements correspond to SecTrace being enabled throughout the duration of the 16 MB TCP transfer (i.e., 16000 packets assuming a 1 KB packet size). In practice, we are likely to turn on SecTrace on a node for only as long as it takes to trace enough packets (say 50 or so) to have a high degree of confidence that

the traced node is receiving all packets. When a multi-hop path is being traced, the duration for which SecTrace is active anywhere along the path increases proportionally to the length of the path. Even so, for realistic path lengths, this duration is likely to be only a fraction of the duration of the 16 MB TCP transfer considered in our experiment.

In summary, we find that SecTrace only imposes a small overhead on performance, and that this overhead is likely to be even smaller in practical settings.

VII. DISCUSSION

In this section, we discuss several attacks on SecTrace and point out how SecTrace can defend against many of these. We also consider action one might take in response to SecTrace’s finding of a suspect link. In particular, we consider routing around the suspect link and present a simple enhancement to the DSR on-demand source routing protocol [13] to enable rerouting in the presence of attackers.

A. Vulnerabilities and Attacks

In the context of wireless mesh networks, we envision SecTrace as contributing one element to the entire task of security management: detecting an inconsistency between a node’s data routing behavior and its route determination behavior in the routing protocol. Although this is a particularly powerful kind of attack on the network, there are other malicious node behaviors that SecTrace is powerless to prevent, and that will have to be dealt with by other components of the mesh network management infrastructure.

For example, a node can behave dishonestly with respect to the routing protocol—pretending, for instance, that available, useful wireless links to neighboring nodes don’t exist. (It might thus be able to free up bandwidth for its own use, that instead would have been used to route other nodes’ traffic.) We do not view such “cheating” as a purely technical problem, since it is in general difficult to tell that a node’s claimed poor or nonexistent connectivity on a link is feigned. Indirect inference based on overhearing the node’s transmissions, say in a wireless network, may be suspect given the significant degree of asymmetry in link connectivity [4]. Furthermore, there may be legitimate nodes, such as those at the edge of a community wireless network, that actually have connectivity to just one neighbor. Thus whether to provide service to such nodes becomes a matter of policy and may call for nodes to be charged for service they consume and credited for service they provide.

SecTrace is also vulnerable to certain attacks, which we discuss here. Our implementation addresses some of these concerns, and solutions to the rest are easy to imagine in the mesh network context.

Implicating the end host: Because the most frequent cause of failed connections will be unresponsive end hosts—a problem which cannot be fixed by routing adjustments—a malicious router can avoid detection via

SecTrace by simulating a “dead” end host, simply by advertising a (non-responsive) direct link to the targeted end host. However, the application of SecTrace in this case would flag the *link* from the malicious router to the end host as suspect. If the claimed last-hop router is very far away from the targeted end host, an attempt to find an alternate route (Section VII-C) should yield one that avoids the offending router. On the other hand, if the malicious router is very close to the targeted end host, then these measures are likely to be less successful; of course, in the worst case, where the misbehaving router is really the (sole) last-hop router, then it will obviously be impossible to distinguish its “blackholing” activity from a truly dead end host. But then, the malicious router can always accomplish the same goal by refusing to acknowledge the link in the first place.

Adjusting behavior to avoid detection: A malicious router may adjust its disruptive behavior so as to avoid detection. For example, it may confine its attacks to periods of time where it does not detect any SecTrace initiation attempts (i.e., key exchange packets from upstream routers).

Section VI-D shows that SecTrace has low enough overhead to be used continually. Thus, doing so removes the danger of a malicious node attempting to disrupt traffic “under the radar”, by watching for packets on the network that appear to be SecTrace packets, and misbehaving only when SecTrace appears to be inactive. The overhead of continual SecTrace can be lowered even further by interleaving “fake” SecTrace attempts (that go through the usual SecTrace handshake to fool adversaries without incurring the overhead of actually tracing packets) with real SecTrace attempts.

Framing innocent nodes: Alternatively, the malicious router may attempt to interfere with SecTrace by selectively blackholing the packets used in the key exchange phase, so as to give the impression that a router further downstream is not accepting key exchanges (and hence either malfunctioning or malicious). This attack cannot be used by a single misbehaving router to frame a router further downstream: if the misbehavior affects normal traffic, then the SecTrace will correctly detect a misbehaving link when the (honest) router immediately downstream of the adversary on the path reports the anomalous traffic pattern. However, as illustrated in Figure 6, two misbehaving routers could collude to frame a router between them on a path; the downstream confederate disrupts traffic, while the upstream one disrupts key exchanges to the victim router so as to implicate it.

A simple countermeasure to this attack (if multiple colluding routers are deemed a threat, and if redundant routes are not being used to effect the key exchange) is to use “onion routing”-style encryption of key exchange messages [33]. Since each step of the traceroute involves a key exchange, the key exchange traffic can be encrypted hop by hop, so that each router along the route does not know the final destination of the message (and therefore cannot consistently frame a single router).

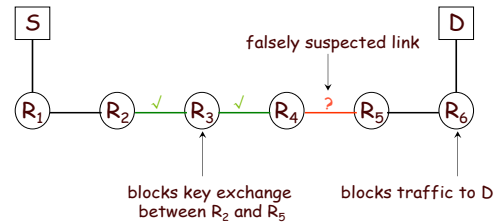


Fig. 6. Misbehaving routers R_3 and R_6 work in conjunction to deceive R_2 into concluding that the link between R_4 and R_5 is faulty.

Our implementation offers a simpler solution to this problem by redirecting all SecTrace control traffic through a trusted ITAP node, protecting against targeted disruption of particular SecTrace attempts. Thus this “framing” attack (also described [20]) will be extremely difficult to mount, since it will be unclear to the adversary which route is being traced by which node or if SecTrace is even active at any point in time. Of course, this procedure would not work if a node is trying to trace the path to the ITAP itself, so we would need to revert to the onion routing style solution in this case.

Creating fictitious nodes: If an attacker can create numerous fictitious nodes in the network, then both identifying a bad link and attempting to route around it can become much more difficult and time-consuming. A single attacker on a path, for instance, could divert SecTrace by returning a bogus next-hop router. This may still remain undiscovered if the bogus router is either a confederate of the attacker or the attacker itself under the garb of a different address (i.e., a “dummy” node). In fact, the attacker could lead SecTrace into a thicket of bogus routers before it peters out. However, SecTrace will eventually identify a bad link, at least one end of which is the attacker or its confederate. Thus, this link deserves to be avoided. There may still be other misbehaving nodes in the path, but persistent application of a succession of SecTrace runs can eliminate them, one by one, from the path until they have all been purged (Figure 7). Hence, if adding confederates or dummy nodes to the network is sufficiently costly, or if owners or administrators of misbehaving nodes are investigated and penalized sufficiently harshly if found to be guilty, then the application of SecTrace would still help identify misbehaving nodes, so that they can be eliminated and/or punished.

However, for our SecTrace implementation we assume a regulated identity infrastructure—in particular, a PKI—to be necessary in any event, to deal with generic DoS threats. Such a PKI would presumably make it difficult for a node to obtain numerous fictitious node identity credentials. Hence this attack is not feasible in our setting.

DoS attack on SecTrace: Finally, the need for computationally expensive key exchanges creates an opportunity for powerful denial-of-service attacks using bogus key exchange messages that require significant computational resources to detect and discard. A simple solution is just

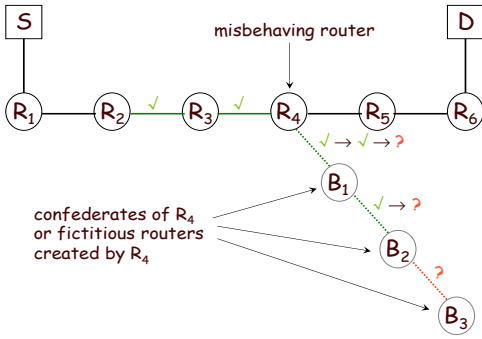


Fig. 7. Misbehaving router R_4 leads SecTrace initiated by R_2 astray, down a path of bogus routers. However, R_2 can still identify and eliminate the bogus links through repeated application of SecTrace.

to throttle the number of SecTrace key exchanges honored to keep it below an acceptable threshold. This raises the possibility of combining malicious routing with denial-of-service attacks to foil SecTrace attempts. One possible solution to this problem is to respond to a key exchange message with a “client puzzle” (as in [32], [34]). Such puzzles are easy for the responding router to generate and check, without maintaining state; the requesting router (or the attacker), in order to have its traceroute request attended to, would have to solve the puzzle—which would require at least the same order of magnitude of computation as the responding router has to perform in order to handle the SecTrace request—and return the solution along with a resend of its key exchange message.

A general traceback facility will presumably be necessary in the mesh network, to deal with DoS attacks. The same facility can also be used to protect against SecTrace-based DoS, (which would take the shape of one node bombarding another with phony SecTrace TLS session requests).

Of course, the attacker could always simply muster the computational resources to mount the attack (say, by harnessing thousands of hacked “zombie” computers to the task). But anyone with that level of resources is likely able to mount a more conventional denial-of-service attack against his intended targets in any event—probably without ever needing to subvert the routing system.

B. Routing Asymmetry

Internet routing is, in general, asymmetric. The path from node A to node B may be different from that from B to A. This asymmetry can create two problems. First, an end host cannot be sure whether its inability to communicate with a peer host is the result of a network problem in one direction, the opposite direction, or both. Second, the same ambiguity can also affect a node that is performing SecTrace. We discuss both these issues in turn.

1) Impact on the End-host Problem Detection: As noted in Section VI-B.2, one of the modes of operation is for end hosts to initiate SecTrace when they detect end-to-end connectivity problems. Consider an end host A that

is trying to communicate with end-host B. If A does not receive any response (e.g., TCP ACKs) to the packets it has sent to B, A cannot be sure whether there is a network problem in the $A \rightarrow B$ direction, in the $B \rightarrow A$ direction, or in both directions. The question then is whether and when end host A should initiate SecTrace.

If A receives a steady stream of packets from B but none that indicates that A’s packets have been received by B, then A can be quite sure that the problem is in the $A \rightarrow B$ direction. Certain applications, such as conferencing, may generate a steady, bidirectional flow of traffic that enables such disambiguation. Certain protocols as well, such as TCP, generate sustained traffic in at least one direction (e.g., TCP retransmissions). In cases where no such traffic is normally generated, peer hosts that are in the midst of communicating with each other could transmit “keep-alive” packets (at a low frequency) during times when they do not have regular packets to send. Receipt of traffic (with little or no loss) from host B would indicate to A that the problem is in the $A \rightarrow B$ direction. A can then initiate SecTrace on the path to B. On the other hand, if the problem is only in the $B \rightarrow A$ direction, then B would hear A’s traffic, deduce that the problem is in the $B \rightarrow A$ direction, and initiate SecTrace on the path to A.

There are, however, two problem cases: (a) there may be a failure in both the $A \rightarrow B$ and $B \rightarrow A$ directions, or (b) the network failure may precede any communication between A and B, preventing the hosts from exchanging any traffic at all. In both cases, since neither A nor B may receive traffic from its peer, neither would be in a position to determine definitively the direction of failure. In such cases, the deadlock can be broken by having the host(s) initiate SecTrace anyway, after having waited for a random extra duration to give its peer the opportunity to initiate SecTrace and possibly resolve the problem.

2) Impact on Secure Traceroute: Asymmetry in network routing can make it difficult for an investigating router, R, to check whether a downstream router, D, is in fact receiving packets. First, even if D is receiving all packets forwarded by R (and hence there is no network problem in the $R \rightarrow D$ direction), the two routers may not be able to establish a secure communication channel simply because of a network problem in the $D \rightarrow R$ direction. Second, even if a secure channel has been established, R may not receive the appropriate response from D due to a (new) network problem in the $D \rightarrow R$ direction. Thus if R’s SecTrace attempt does not elicit the appropriate response from D, R cannot be sure whether there is a problem in the $R \rightarrow D$ direction or in the $D \rightarrow R$ direction.

The solution we suggest is as follows: The investigating router, R, first initiates SecTrace as described earlier. If it does not receive the appropriate response from a certain downstream router, D, router R repeats SecTrace with one difference: it includes the reverse route that D could use to communicate back to R using a mechanism such as IP source routing. The reverse route consists of the sequence

of intermediate routers along the path from R to D in reverse order. (Note that due to routing asymmetry, the reverse route may not be the same as the route from D to R.) The underlying assumption is that if in fact there is no network problem in the R→D direction, it is quite likely (modulo the presence of unidirectional links) that D will be able to communicate back to R via the reverse route. Although wireless links are known to be asymmetric [4], such asymmetry is often in terms of performance characteristics rather than connectivity. So the SecTrace response can typically still traverse the reverse path from D to R.

In the worst case, the inability of D to communicate back to R would just mean that R would incorrectly deduce that the problem is at or around D and would proceed with unnecessary rerouting or other corrective action, an undesirable but hardly disastrous outcome.

3) Impact of Redirection via the ITAP: The foregoing discussion assumed direct communication between the investigating router, R, and the investigated router, D. However, as noted in Section VI-B.3, our implementation in the wireless mesh network redirects all control messages between R and D via the ITAP. On the one hand, this means that asymmetry in the network path in the R→D and D→R directions will not present hurdles to SecTrace. On the other hand, redirection via the ITAP introduces extraneous failure modes due to problems on the path to/from the ITAP. However, communication via the ITAP is just an optimization, and we can always fall back on direct communication between R and D should either of the nodes be unable to communicate with the ITAP. Separately, SecTrace can be used to investigate the path to/from the ITAP, with the asymmetry-related precautions discussed earlier.

C. Responding to the Findings of SecTrace

Since connectivity problems are often likely to be due to non-malicious causes (congestion, link failure, etc.), a reasonable first step would be to route around the suspect link. If repeated attempts at rerouting are of no avail, or worse, if SecTrace continues to flag the same link or node as suspect, then it becomes more likely that an attacker is responsible for the connectivity problem. It would then make sense to investigate the problem, possibly with human intervention, and either rectify the faulty router or eject it from the network—say, by revoking its credentials.

When a node runs SecTrace and finds a link to be suspect, it may be the only node in the network that knows about (or is even affected by) the suspect link. It is possible for an attacker to interfere with and effectively block the lone node’s attempt to route around the suspect link. Since other nodes in the network may not see the need to avoid the suspect link, an attacker who is strategically placed along the shortest path may be hard to route around.

Figure 8 illustrates an example in the context of an on-demand source routing protocol such as DSR [13] used in

an ad hoc wireless network setting. Node A suspects link $X \rightarrow Y$ of being faulty and wishes to find a route to F that avoids this link. Such a route does exist (through $B \rightarrow C \rightarrow D$) but is longer in terms of hop count. When A floods its route-request, the copy of the message that traverses the path through $X \rightarrow Y$ is likely get to E (the point at which the two paths converge) first and hence be forwarded onto F . When the copy of the route-request arrives via the path through $B \rightarrow C \rightarrow D$, E will consider it to be a “duplicate” and hence suppress it. So A will be unable to discover the alternate path via $B \rightarrow C \rightarrow D$.

As noted above, the inability to route around the suspect link despite repeated attempts could trigger human intervention and eventual resolution of the problem. However, since this is likely to be time-consuming, it would be desirable to have the ability to reroute effectively. The key is for the node that is trying to find a new route (A in Figure 8) to make other nodes in the network (E in particular in Figure 8) aware of the identity of the suspect link(s) or node(s). (Note that we cannot be sure which end of the suspect link is faulty, as explained in Section III-A, so routing around the link can either mean avoiding both ends of the link or avoiding just one end initially.) For a protocol such as DSR, this can be accomplished by augmenting the route-request message with an *exclusion list* identifying the link(s) or node(s) that the originator wishes to avoid. (The originator would have to sign the exclusion list to assure its authenticity, thereby preventing attackers from tampering with the list.) Other nodes in the network would then take the exclusion list into account when deciding whether to forward or suppress a route request. For instance, in the scenario depicted in Figure 8, $X \rightarrow Y$ would be included in the exclusion list of the route-request broadcast by A . So E would suppress the copy that arrives via $X \rightarrow Y$ and forward the one that arrives via $B \rightarrow C \rightarrow D$ even if the former arrives first.

While it is incumbent on nodes to honor the exclusion list specified in a route-request message, they should not use this information to alter their own route discovery attempts. Otherwise, it would open the door for a node to launch an attack by framing innocent nodes.

VIII. RELATED WORK

There have been several pieces of work on making Internet routing protocols robust against attacks by malicious entities. The earliest comprehensive treatment of the problem to our knowledge was by Perlman [20], [21], who presented an approach for “sabotage-proof” routing in the presence of Byzantine node failures. The key idea is to use “robust flooding” to distribute link-state packets (LSPs) and the public keys of all nodes throughout the network. Robust data routing is then accomplished by having end hosts construct digitally signed source routes using the link-state information they have gathered. If the chosen source route fails, node-disjoint alternative routes are tried. In the extreme case, data packets are distributed via robust

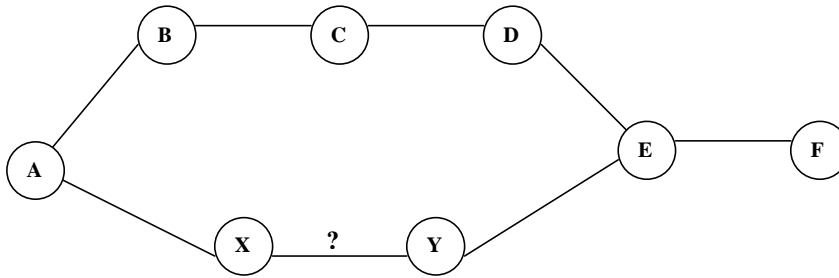


Fig. 8. Illustration of how an attacker such as X or Y on the shortest path could frustrate A 's attempt to discover an alternate path to F in the context of an on-demand source routing protocol such as DSR.

flooding. While this scheme is very robust in theory — a working route will be found if it exists — there are obvious practical hurdles to its use. For instance, flooding the network with data packets is likely to be expensive, perhaps even more so than the disruption caused by the failed routers.

Perlman also discusses a strategy for fault isolation that involves having intermediate routers return ACKs for specially marked packets. However, she deprecates this scheme in favor of simply finding node-disjoint alternative paths because of the former's space and computational complexity. She also points out that two colluding faulty routers could fool the fault isolation scheme by framing innocent routers in between. The SecTrace scheme we present in this paper shares the philosophy of Perlman's fault isolation scheme. However, as discussed in Section VI-D, SecTrace maintains efficiency by doing signalling out of band rather than requiring in-band ACKs. Also, as explained in Section VI-B.3 and Section VII-A, SecTrace is resistant to the “colluding routers” attack.

Other proposals [15], [23] have considered less radical approaches to securing the routing protocol. In particular, Secure BGP (S-BGP) [15] proposes using public key infrastructures (PKIs) and IPSec [14] to enable a BGP speaker to validate the authenticity and data integrity of BGP UPDATE messages that it receives and to verify the identity and authorization of the senders. However, authentication of routing information does little to help detect or diagnose faulty routing information emanating from a router or to protect against a misconfigured or failed router that is authorized to advertise routes for an address prefix but fails to deliver packets anyway. Moreover, it does nothing to protect against routers that obey the routing protocol perfectly, but misbehave when actually forwarding data.

A recent piece of work [24] has tried to address the problem of routing misbehavior in both the control plane and the data plane in the context of BGP. A “whisper” procedure is used to identify inconsistencies in route advertisements received for a particular destination. A “listen” procedure is used to identify non-working routes by monitoring end-to-end TCP flows. The Listen and Whisper (L&W) solution

differs from SecTrace in both the assumptions it makes and its goals. L&W does not depend on a PKI, but it is unable to identify faulty routers or links and is also susceptible to attacks on the implicit feedback that “listen” depends on. While L&W is a useful solution, we believe that it is inadequate for an open network setting where routers can be arbitrarily malicious.

There has been considerable interest recently in securing routing in mobile ad hoc networks. In [18], a “watchdog” technique is proposed to enable a node to check that a neighboring node did in fact forward a packet onward without tampering with it. This technique makes the strong assumption that nodes can hear the onward transmissions of their neighbors, which may not hold even in wireless ad hoc networks (for instance, due to local interference, directional antennae, and the asymmetry of wireless links [4]). For instance, the onward transmission might use an aggressive modulation scheme corresponding to a good quality channel (as in 802.11 multi-rate schemes) and hence may not be decodable by nodes in the vicinity that have a poor channel from the transmitter. Note that the assumption of symmetry made by the watchdog technique is much stronger than made by the path-reversal heuristic discussed in Section VII-B.2. The former requires symmetry in channel characteristics whereas the latter only depends on symmetry in connectivity.

SEAD [9] focusses on a lightweight scheme to enable nodes to authenticate routing updates from other nodes for the specific case of a distance-vector routing protocol. As with S-BGP, authentication does not solve the problem of a faulty node that fails to forward packets.

Ariadne [10] proposes a similar solution in the context of an on-demand routing protocol, DSR [13]. Ariadne addresses the problem of forwarding misbehavior by maintaining multiple (node-disjoint) source routes to the destination and adapting the fraction of traffic sent on each route based on end-to-end feedback on packet delivery success rate. However, a significant drawback of this approach is that a single attacker could force the sender to avoid all (good) nodes along the affected route(s), resulting in unnecessary performance degradation. Rather than “shoot in the dark”, our approach is to use SecTrace to localize

the fault so that it can be avoided with minimal rerouting.

[11] proposes a self-organized PKI suitable for mobile ad hoc networks. In the absence of a centralized PKI, we could use a similar approach to support secure traceroute.

IX. SUMMARY

In this paper, we have argued that securing routing in open networks calls for more than just steps to secure the routing protocol — it is also important to secure packet forwarding. To this end, we have described a secure traceroute protocol designed to detect and localize the cause of packet forwarding misbehavior inconspicuously, thereby denying attackers the opportunity to mislead the investigation. We have applied SecTrace in the context of a community wireless network and have discussed domain specific choices made in our implementation. Our experimental results indicate that SecTrace imposes a negligible performance overhead in practice, making it suitable for use in routine monitoring of end-to-end paths in an open network. We have considered attacks on SecTrace and presented ways of defending against many of them. We have also considered the problem of routing around link(s) flagged as suspect by SecTrace and presented a simple enhancement to the DSR protocol to enable such rerouting in the face of attackers.

REFERENCES

- [1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. “Resilient Overlay Networks”, *ACM SOSP*, October 2001.
- [2] M. Bellare and P. Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”, *ACM CCS*, 1993.
- [3] M. Bellare and P. Rogaway. “Optimal Asymmetric Encryption”, *Eurocrypt '94*, LNCS 950, Springer-Verlag, pp. 92–111, 1995.
- [4] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. “A High-Throughput Path Metric for Multi-Hop Wireless Routing”, *ACM Mobicom*, September 2003.
- [5] D. Dean and A. Stubblefield. “Using Client Puzzles to Protect TLS”, *USENIX Security*, August 2001.
- [6] T. Dierks and C. Allen. “The TLS Protocol Version 1.0”, *RFC-2246*, *IETF*, January 1999.
- [7] J. R. Douceur. “The Sybil Attack”, *IPTPS*, March 2002.
- [8] R. Draves, J. Padhye, and B. Zill. “Comparison of Routing Metrics for Static Multi-Hop Wireless Networks”, *ACM SIGCOMM*, August 2004.
- [9] Y. C. Hu, D. B. Johnson, and A. Perrig. “SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks”, *IEEE WMCSA*, June 2002.
- [10] Y. C. Hu, A. Perrig, and D. B. Johnson. “Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks”, *ACM Mobicom*, September 2002.
- [11] J. P. Hubaux, L. Buttyan, and S. Capkun. “The Quest for Security in Mobile Ad Hoc Networks”, *ACM MobiHoc*, October 2001.
- [12] V. Jacobson, “The Traceroute Manual Page”, Lawrence Berkeley Laboratory, Berkeley, CA, USA, December 1988.
- [13] D. B. Johnson and D. A. Maltz. “Dynamic Source Routing in Ad hoc Wireless Networks”, In *Mobile Computing*, Chap. 5, pp. 153-181, Kluwer Academic Publishers, 1996.
- [14] S. Kent and R. Atkinson. “Security Architecture for the Internet Protocol”, *RFC 2401*, November 1998.
- [15] S. Kent, C. Lynn, and K. Seo. “Secure Border Gateway Protocol (Secure-BGP)”, *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 4, April 2000.
- [16] J. Kohl and C. Neuman. “The Kerberos Network Authentication Service (V5)”, *RFC-1510*, *IETF*, September 1993.
- [17] H. Krawczyk, M. Bellare, and R. Canetti. “HMAC: Keyed-Hashing for Message Authentication”, *RFC-2104*, *IETF*, February 1997.
- [18] S. Marti, T. J. Giuli, K. Lai, and M. Baker. “Mitigating Routing Misbehavior in Mobile Ad Hoc Networks”, *ACM Mobicom*, August 2000.
- [19] V. N. Padmanabhan and D. R. Simon. “Secure Traceroute to Detect Faulty or Malicious Routing”, *ACM HotNets*, October 2002.
- [20] R. Perlman. “Network Layer Protocols with Byzantine Robustness”, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, MIT, August 1988.
- [21] R. Perlman. “Interconnections: Bridges, Routers, Switches, and Internetworking Protocols”, *Addison-Wesley Professional Computing Series*, Second edition, 1999.
- [22] A. Shamir. “How to share a secret.” *Communications of the ACM*, 22:612–613, 1979.
- [23] B. R. Smith and J. J. Garcia-Luna-Aceves. “Securing the Border Gateway Routing Protocol”, *Global Internet*, November 1996.
- [24] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. “Listen and Whisper: Security Mechanisms for BGP”, *USENIX/ACM NSDI*, March 2004.
- [25] “Advanced Encryption Standard (AES)”, *U.S. Federal Information Processing Standards Publication 197 (FIPS-197)*, November 2001.
- [26] Bay Area Wireless Users Group, <http://www.bawug.org/>
- [27] Home Phonenumber Networking Alliance (HomePNA), <http://www.homepna.org/>
- [28] HomePlug Powerline Alliance, <http://www.homeplug.com/>
- [29] Seattle Wireless, <http://www.seattlewireless.net/>
- [30] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas and T. Ylonen. “SPKI Certificate Theory”, *RFC 2693*, September 1999.
- [31] P. Zimmermann, “The Official PGP User’s Guide”, MIT Press, 1995.
- [32] C. Dwork and M. Naor. “Pricing Via Processing or Combatting Junk Mail”, In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 139-147, 16-20 August 1992. Springer-Verlag, 1993.
- [33] P.F. Syverson, G. Tsudik, M. G. Reed, and C. E. Landwehr. “Towards an Analysis of Onion Routing Security”. In H. Federath, editor, *Designing Privacy Enhancing Technologies*, vol. 2009 of LNCS, pp. 96–114. Springer-Verlag, 2001.
- [34] A. Juels and J. Brainard. “Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks”, *NDSS '99 (Networks and Distributed Security Systems)*, February 1999.