Trusted Application-Centric Ad-Hoc Networks *

Gang Xu[1,3], Cristian Borcea[2], and Liviu Iftode[1]
[1] *Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA*
[2] *Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA*
[3] *AT&T, 200 Laurel Ave, Middletown, NJ 07748, USA*
{gxu, iftode}@cs.rutgers.edu, borcea@cs.njit.edu

Abstract

Nodes in MANETs lack the protection offered by firewalls in infrastructure-based networks because malicious nodes can roam into the vicinity of another node and start launching attacks. This paper presents a distributed mechanism that allows trusted nodes to create protected networks in MANETs. A protected network is created to run a specific application and enforce a common network access control policy associated with that application. To become a member in the protected network, a node has to demonstrate its trustworthiness by proving its ability to enforce policies. Attacks from untrusted nodes are impossible because these nodes are not allowed to establish wireless links with member nodes. Attacks from member nodes are stopped at the originators by the network policy. The trusted execution of all programs involved in policy enforcement is guaranteed by a kernel agent. We demonstrate the correctness of our solution through security analysis and its feasibility through a prototype implementation tested over an IEEE 802.11 ad hoc network.

1 Introduction

With the appearance of a large number of laptops, smart phones and wireless-enabled vehicular systems, applications running over mobile ad hoc networks (MANETs) can finally make their real-life debut [20, 1]. However, many people are still reluctant to allow their devices to join MANETs because current technologies can at most react to attacks from malicious users after these attacks happened. Unfortunately, they cannot prevent attacks from reaching "good" nodes as firewalls do in infrastructure-based networks. Practically, the concept of protected networks, shielded from external attacks by a firewall enforcing a network access control policy, does not exist in MANETs. The reason is that malicious nodes can simply roam into the wireless transmission range of another node, establish a direct wireless link, and start launching attacks.

As an example, let us consider a group of users who run a peer-to-peer file sharing application across a MANET composed of their smart phones. Through a direct wireless link or ad hoc routing, a malicious user can attempt to exploit a vulnerability in this application to compromise other nodes. Since there are no prior trust relationships between the nodes, it is impossible to identify the untrusted nodes and protect against them. Furthermore, even if these nodes are known and the attacks launched from them are detected, these attacks can still reach the trusted nodes causing depletion of resources on smart phones (e.g., battery). To make things even worse, given the relatively low network capacity, a single attacker can flood the entire network and make it unusable.

This paper presents a distributed mechanism to create protected MANETs. Our solution is driven by an application-centric view of MANETs. In most cases, a group of nodes form such networks to execute a specific application (e.g., a game, a peer-to-peer file sharing application). Therefore, it is possible to define a common policy agreed upon by all participants who want to benefit from the application and foil attacks. The right policy and enforcing software (referred to as *enforcer*) must be in place in order to stop a specific attack. In their simplest forms, a policy can be defined as a set of access control rules, and the enforcer can be a regular packet filter. Each participant must obtain the policy before joining the protected network. The initial nodes of the network obtain the policy through negotiation or from the authority that deploys the network. New participants can download it from other member nodes.

Our mechanism requires all member nodes to *enforce* the common network access control policy, specific to the application that triggered the formation of the network. A node's trustworthiness in enforcing the policy is verified before it can establish link layer connectivity with the protected network. Attacks at the network or above layers from

*This work was supported in part by the NSF grants CCR-0133366, ANI-0121416, CNS-0520123 and CNS-0520033

1-4244-1455-5/07/$25.00 ©2007 IEEE

untrusted external nodes are impossible because these nodes cannot establish wireless links with any member node. Attacks from trusted [1] member nodes are suppressed at the originators by the common network policy. The trusted enforcement of the policy is guaranteed by a kernel agent, which ensures that a node can communicate in the network only if the execution of all programs involved in policy enforcement is not tampered with; otherwise, the agent tears down the links of its node.

This mechanism is built on top of Satem [28], our trusted computing system based on a low-end trusted hardware, the Trusted Platform Module (TPM) [26], which guarantees trusted execution for a set of software components. Due to its low cost and broad support by computer makers, the TPM has been already integrated in many laptops (e.g., HP nc6020, Lenovo ThinkPad T43). In the near future, it will also be installed on smaller mobile devices such as PDAs and mobile phones [27], which makes the TPM-based approach usable for many types of ad hoc networks. We implemented a prototype on Linux-based systems and tested it over an IEEE 802.11-based ad hoc network. The experimental results demonstrate the feasibility and low overhead incurred by our system.

The paper is organized as follows. Section 2 gives a brief overview of Satem, our trusted computing system. In Sections 3 and 4, we describe the system architecture and the security protocols that support trusted policy enforcement in ad hoc networks. We demonstrate the use of our method through a case study in Section 5. Section 6 presents the prototype implementation and its experimental evaluation. Section 7 addresses issues regarding multiple applications, policy update and mobility. Related work is discussed in Section 8. Finally, the paper concludes in Section 9.

2 Satem: A Trusted Computing System

This section presents an overview of Satem [28], our trusted computing system that was leveraged to build the mechanism for creating and maintaining trusted ad hoc networks. Originally, we designed and implemented Satem to ensure requesters of a remote network service that the service executes only trusted code.

Overview. Satem is composed of a *trusted agent* in the OS kernel of the service platform and a *trust evaluator* on the user platform. The service provider performs the attestation of the OS kernel (including the trusted agent) through a trusted boot process using the TPM specified by the Trusted Computing Group (TCG). Subsequently, the trusted agent takes advantage of the service execution context to attest only the code loaded dynamically by the service. More importantly, it ensures that the service executes only trusted code by protecting the service execution in the OS kernel.

Central to Satem is the commitment protocol. Before starting a transaction with a service, users request the trusted agent to provide the attestation of the OS kernel, a *kernel commitment*, and a *service commitment*. The commitments describe all the code files the kernel and the service may execute in all circumstances (e.g., executables, libraries). The trusted agent enforces the kernel commitment at boot time and the service commitment upon being started, such that the kernel and the service are forbidden to load any code files that are either undefined in the commitment or tampered with. Therefore, if the requester verifies that the kernel, the agent, and the commitments are trusted, it is convinced that (1) the service has executed only trusted code up to the time of attestation; and (2) the service will continue to do so in the following phases due to the protection provided by the trusted agent.

Trusted System Initialization. In Satem, the first step is to establish the trusted computing base that includes the trusted agent and the entire OS kernel. This process involves a trusted boot, in which each component in the boot sequence, starting from the TPM, attests the next one before handing over the control. For instance, on a Linux system such as the one used for our prototype, TPM computes the `SHA1` hash over the BIOS image, BIOS computes the `SHA1` hash over LILO, and LILO does the same for the OS kernel. The attestation result is saved in a TPM internal register and can be retrieved as a TPM *report*, which is signed with a TPM internal private key to prove its genuineness. The trusted agent is loaded along with the kernel. The user is not allowed to disable the agent unless she reboots the machine. As a result, the TPM report containing the attestation result saved in the TPM is sufficient to prove a genuine kernel and trusted agent.

Commitment. Satem defines one kernel commitment for the system and one service commitment for each service that wants to use it (referred to as protected service). Each piece of software described in the commitments is defined by a combination of its identifier (e.g., name and version) and the `SHA1` hashes of all its code files. To verify the commitment, the requester must have the correct `SHA1` hashes of all the code files defined in the commitment; but given the wide variety of software, this is impractical. We solve this problem by generating the commitment as a certificate through cooperation with the service software vendors and a third-party trusted certificate authority (CA). This process consists of two steps:

1. *Request code certificates.* The service provider requests each vendor to generate a self-signed code certificate in the same format as the commitment for its code.

[1] By trusted node, we mean any node that can be trusted to enforce the common network access control policy, independent of its trusted or untrusted owner.

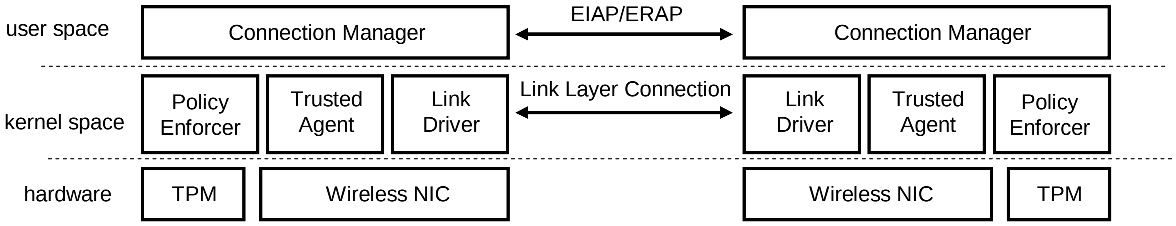

Figure 1. The Common Security Architecture at Nodes

2. *Sign the commitment.* The requester forwards all the code certificates and the commitment to a third-party trusted CA. The CA needs to verify the signatures of all code certificates and compare the code hashes in the commitment against the certificates. The CA signs the commitment if and only if it verifies all code certificates and code hashes in the commitment.

Satem only guarantees the integrity and the authenticity of the code, but not its correctness. The requester must have a local trust policy that governs which kernel and services are trusted. It takes two steps to verify whether a service is trusted. First, it authenticates the kernel and service commitment certificates and learns the identities of the kernel, its modules, and the service [2]. Second, it verifies the kernel and the service against the trust policy.

Enforcement The agent enforces the kernel and service commitments in the same way. It inserts checkpoints in the kernel functions related to new code loading, such as `do_execve`, `sys_init_module`, `do_read` and `do_mmap`, etc. Each checkpoint does the following:

1. checks if the current process belongs to the protected service;
2. if positive, attests the code to be loaded and compares the result with the commitment;
3. if they do not match, the attempted load is denied;
4. if they match, the trusted agent loads the code and protects it in memory by standard process isolation provided by the OS kernel. In particular, it disables direct memory access from user space via `/dev/mem` and `/dev/kmem`.

3 Security Architecture

The essential idea of our mechanism is to allow only nodes that can be trusted to enforce a common network policy to be part of the protected ad hoc network, regardless of whether the node is owned by a trusted entity or not. To accomplish this goal, each node supports a modified version of Satem (i.e., the *trusted agent* is modified for policy enforcement) augmented with a *connection manager* and a *policy enforcer* as illustrated in Figure 1. The connection manager establishes the link layer connection and shares the policy with other nodes. The policy is a set of access control rules, and the enforcer is a packet filter, such as Linux netfilter, that can enforce the rules.

We adapt Satem to ensure trusted execution of all code involved in the policy enforcement. First, according to Satem, the OS kernel (including the trusted agent, the policy enforcer, and the link layer driver) is attested by default. Second, we define a *link commitment*, which includes the connection manager in its protection scope. Third, we modify Satem's service-oriented commitment protocol and integrate it into the link layer access control protocols. This part of the design will be detailed in Section 4.

Commitment Evaluation. In Satem, evaluating a commitment is reduced to authenticating the certificate. In general, this is non-trivial in ad hoc networks due to the lack of constant connection to the Internet and PKI. A node may still be able to authenticate a certificate if it locally holds the public key of the signing certificate authority or a valid certificate chain to it. However, it is unable to validate in real-time the certificate since it has no access to the CA's certificate revocation list.

The above issue can be significantly alleviated given the special nature of the problem we aim to solve. As discussed in [10], although nodes do not have persistent Internet connectivity, they can still get online from time to time. For example, a user may be off-line on an inter-city train most of the time, but get online when the train enters a train station. Furthermore, a Satem commitment only states that a code file has a certain corresponding `SHA1` digest. This fact is invariant under any circumstance. Lastly, since the ad hoc network is formed for a specific task and only lasts for a relatively short period of time, the likelihood of revoking a certificate is negligible.

Based on the above observations, we introduce a short-life certificate to authenticate the commitment. When being connected to the Internet, each node obtains a regular long-

[2] The commitment authentication is not trivial in ad hoc networks. Section 3 presents our solution for such networks.

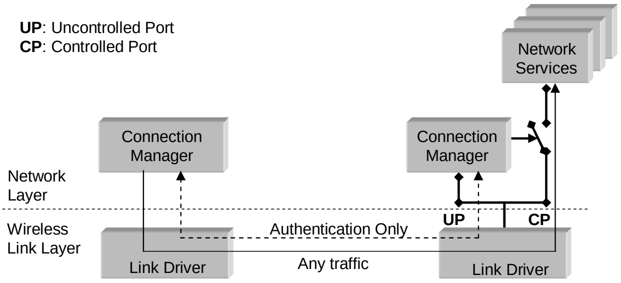

Figure 2. Dual Port Access Control

life commitment certificate C_L, a short-life commitment certificate C_S and the authority's certificate C_A. When losing Internet connectivity, the node can still use the C_A to authenticate C_S of other nodes. Since this certificate is only good for a short period of time, there is no need to be concerned about revocation. After C_S expires, the node needs to regain Internet access to renew it using its C_L. The CA verifies the C_L using PKI and grants the renewal request without re-authenticating it from scratch.

Policy Creation. If a MANET is deployed by an authority, this authority is responsible for defining and updating the policy. To ensure its authenticity, the authority signs the policy and preloads its public key on each node. If the MANET is created spontaneously by a set of nodes (referred to as the creators of the network hereafter), the creators negotiate the policy using existing links. A typical scenario is that the node that initiates the network, drafts the policy and proposes it to the others. The policy can be signed using threshold based methods such as [16, 29]. When the policy is defined, all creators drop the existing wireless links with each other. These peers then rejoin the network by running the secure protocol described in Section 4.

In spontaneously-created MANET, there is no assumption about the correctness or trustworthiness of the policy. All the creators may be malicious, and they form the network just to attract innocent nodes. Therefore, a node may need to evaluate the policy independently based on its own need and capability of enforcing the policy. Since all nodes intending to join the network have the goal of executing a common application, they are motivated to accept a reasonable common access control policy to benefit from the application and protect themselves from attacks.

Each node may already have a local policy to protect itself from malicious nodes. There may be a conflict between the local policy and the network policy. Policy rule conflict discovery and resolution has been extensively studied and many methods are available [9, 18]. In this paper, we assume the enforcer is able to use one of these methods.

4 Protocols

Our approach requires that the link layer have the following properties:

P1: *dual port access control*;

P2: *secure link association*;

The concept of dual port access control is defined in IEEE 802.1x [4], which is supported by IEEE 802.11 (e.g., Windows XP and Linux provide such support). The link layer of a network node has two access ports, uncontrolled and controlled, as shown in Figure 2. The controlled port has full access to the link layer, while the uncontrolled port is used only for authentication. Any node in the transmission range can connect to the uncontrolled port in order to authenticate itself. Only after a successful authentication and authorization by the peer node, the initiator can connect to the controlled port.

Once a connection is established, the link layer must ensure that it cannot be hijacked. This is addressed by secure link association. In practice, **P2** is accomplished using link layer encryption (i.e., all frames transmitted over the link are encrypted). Consequently, a node with a single wireless card can be part of only one protected MANET at a time.

We define two protocols that allow a node to join a protected network: Enforcement Initial Activation Protocol (EIAP) and Enforcement Re-Activation Protocol (ERAP). Both protocols perform three tasks: (1) verifying the mutual trustworthiness of nodes, (2) sharing the link layer key, and (3) distributing the network policy. EIAP is used to join a network for the first time, while ERAP is used to re-join a network; their main difference is the verification method.

Assuming that a new node N_e wants to join a trusted ad hoc network AN by initiating a link layer connection request to a member node of AN, N_m, the high-level view of our security protocols is as follows:

1. N_e calls its connection manager to establish a link layer connection to N_m. The connection manager of N_m grants the connection only to its uncontrolled port.

2. The connection manager of N_e calls the trusted agent to attest the capability and trustworthiness of enforcing AN's policy.

3. The connection manager of N_m verifies the attestation, attests itself and pushes AN's policy P and the link layer key k to N_e. It then grants N_e full access to its controlled port.

4. After verifying N_m's attestation, the connection manager of N_e invokes the policy enforcer to enforce P and finalizes the connection with the link layer key k. Note that the enforcer must be already loaded in the OS kernel.

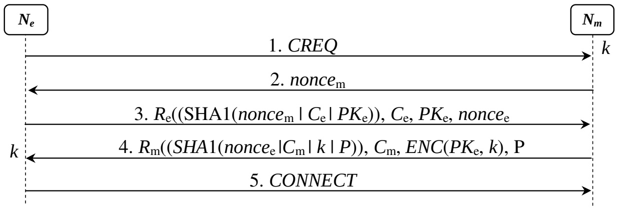
N_e
N_m
1. $CREQ$
k
2. $nonce_m$
3. R_e((SHA1($nonce_m$ | C_e | PK_e)), C_e, PK_e, $nonce_e$
k
4. R_m(($SHA1$($nonce_e$ |C_m | k | P)), C_m, ENC(PK_e, k), P
5. $CONNECT$

Figure 3. The Enforcement Initial Activation Protocol (EIAP)

In the rest of the section, we present the two protocols in detail, followed by a discussion of the limitations of our approach.

4.1 Enforcement Initial Activation Protocol (EIAP)

When a node N_e that has never connected to any node of AN attempts to connect to a member of AN, N_m (we say N_e connects to AN via N_m), N_m activates the enforcement of AN's policy P on N_e through the EIAP protocol. The protocol is illustrated in Figure 3.

1. The connection manager of N_e initiates a wireless connection request $CREQ$ to N_m.

2. The connection manager of N_m grants access to its uncontrolled port and replies with a random number, $nonce_m$.

3. The connection manager of N_e attests itself and delivers the commitments to N_m. To attest itself, the connection manager of N_e calls the trusted agent to generate a TPM report (R_e) of the kernel attestation results with $SHA1(nonce_m|C_e|PK_e)$ as the parameter ($|$ means concatenation). C_e represents the link commitment and the kernel commitment. PK_e is the public key generated by the connection manager. The TPM signs both the parameter and the attestation results. N_e then generates another random number $nonce_e$ and sends R_e, C_e, $nonce_e$ and PK_e to N_m

4. The connection manager of N_m activates the enforcement of P on N_e. The connection manager of N_m validates R_e to make sure it is generated by a valid TPM. Then, it validates the parameter by recomputing the hash using its stored $nonce_m$ and the newly received C_e and PK_e. Next, it generates a TPM report in the same way as N_e with a parameter $SHA1(nonce_e|C_m|k|P)$, where C_m represents the link and kernel commitments of N_m, k is the link layer session key, and P is the network access control policy. Finally, the connection manager encrypts the link layer session key k with PK_e, $ENC(PK_e, k)$ and sends it with R_m, C_m, and P to N_e. Then, N_m grants N_e full access to its controlled port.

5. N_e establishes a full connection with N_m. N_e validates R_m the same way as N_m did. Then, it evaluates P before accepting it. Once P is accepted, it is pushed to the policy enforcer, which starts enforcing it immediately. Finally, N_e decrypts k using the corresponding private key SK_e, enables link layer encryption, and obtains full connectivity to N_m's controlled port.

The EIAP protocol enables the two nodes to mutually verify trustworthiness in enforcing the policy. This is necessary because the link to be established is bi-directional and both nodes need to protect themselves from each other.

Security Analysis. Let us consider a local attacker on N_e (the analysis holds if the attacker is on N_m). We assume that the attacker cannot break the TPM or launch hardware based attacks, and in particular, cannot use direct memory access (DMA). We further assume that the attacker is unable to bypass the node operating system to gain access to system resources such as memory, CPU, network card, and disk. Other than these restrictions, the attacker can have full control of the software system, including superuser privileges.

Disable enforcement of P. The most direct attack is to disable the enforcement of P after obtaining the connectivity. The attacker can do so by disabling the policy enforcer. This requires removing the policy enforcer's kernel module. The trusted agent intercepts the removal request, clears k, and tears down the link before removing the module. Thus, the attacker has to first disable the trusted agent. As we discussed earlier, once being turned on, the trusted agent cannot be turned off until next reboot.

Modify P. The attacker may attempt to modify the current policy P at runtime. The trusted agent secures the memory space holding P such that only the connection manager have write permissions to it. Additionally, the connection manager is protected by the trusted agent. The attacker may try to run a malicious connection manager. This is allowed only if it is described in the link commitment, which means the link commitment is also untrusted. N_m will receive the commitment at step 3 and refuse to trust the malicious connection manager.

Steal k. The attacker may attempt to steal the session key k on the node. The key is secured by the trusted agent in memory and accessible only to the connection manager and the link driver [3]. The protocol ensures secure distribution of the key. On one hand, the key owner will not distribute the key to any untrusted node (step 4). On the other hand,

[3] Some drivers provide simple user space utilities for users to read the link layer session key. In our method, these utilities will fail.

a node that joins the network will not accept the key from a member node unless the member node has been verified to be trusted (step 5). Consequently, an untrusted node cannot create a key and fool others to accept it.

Hijack k. The attacker may try to steal k in distribution. The public-private key pair is dynamically generated, and the private key SK_e is saved in the connection manager's memory and never disclosed to any other processes. The trusted agent protects the key from being disclosed to any process other than the connection manager. Hence, the attacker is unable to acquire SK_e to decrypt k intercepted at step 4.

Play man-in-the-middle. The attacker may attempt to play a man-in-the-middle attack by replacing PK_e with her own PK_a in order to decrypt k with the private key SK_a she owns. Although PK_e is not authenticated in the protocol, it is attested in the report R_e. The TPM report certifies that the public key PK_e belongs to the system attested by the TPM. Therefore, unless the attacker's system is fully trusted, which makes it impossible to launch attacks, N_m will detect this and refuse to distribute k. The attacker may want to replay a valid TPM report and exploit it to gain trust from N_m. The protocol foils the attacker by including $nonce_e$ and $nonce_m$ in the attestation reports.

4.2 Enforcement Re-Activation Protocol (ERAP)

EIAP addresses the scenario where an external node N_e connects to AN via $N_m \in AN$ for the first time. In MANETs, the connectivity between nodes and the network topology may change constantly. For instance, N_e may roam out of the wireless transmission range of N_m, and thereby, it will lose the previously established connection. Subsequently, it may approach another node $N_h \in AN$ and re-establish connection to AN via N_h. This brings up two issues, which make the use of EIAP unsuitable for connection re-establishment.

First, the network policy may be updated to accommodate certain changes in the network, which causes the local copies on N_h and N_e to be inconsistent. We need to ensure that only the most recent version of P is enforced. This problem can be solved by assigning a version number v to P, which is incremented every time P is updated. Second, the nodes enforcing old policies should be disconnected from the network. This can be done by having the nodes enforcing the new policy update their link layer session key. As a result, when the node rejoins the network, it has to request the new key to re-establish the connectivity.

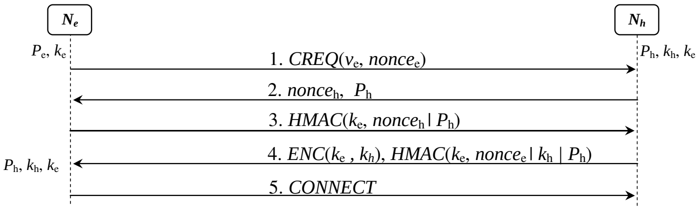

Figure 4. The Enforcement Re-Activation Protocol (ERAP)

EIAP can solve the above problems, but it is too costly due to the need to generate the TPM report and transmit large data via a partial link. Therefore, it is desirable to optimize the protocol for the link re-establishment scenario. We observe that both N_e and N_h must have been verified before they were allowed to connect to AN for the first time. Owning an old key implies that it has been protected by the trusted agent. Otherwise, if any program defined in the commitments was compromised since last connection, the trusted agent would have wiped out the key. Therefore, we can let each node hold the past V keys and use them to compute a keyed message authentication code (HMAC) to prove its trustworthiness. Figure 4 illustrates the Enforcement Re-Activation Protocol (ERAP) assuming N_h has the latest policy.

1. N_e sends a request to N_h to establish a full link. N_e includes its policy version v_e and a random number $nonce_e$ in the request. N_e cannot lie about its v_e since it will be verified later by N_h.

2. N_h grants access to its uncontrolled port. N_h replies with another random number $nonce_h$. It also compares its policy version v_h with v_e and includes the latest policy P_h in the reply if $v_h > v_e$.

3. N_e authenticates to N_h. If N_e authenticates P_h, it learns that its policy needs to be updated. It computes $HMAC(k_e, nonce_h|P_h)$ and sends it back to N_h.

4. N_h verifies N_e and distributes k_h to N_e. Verifying $HMAC(k_e, nonce_h|P_h)$ proves that N_e holds k_e and P_h. N_h then encrypts k_h with k_e ($ENC(k_e, k_h)$) and sends the encrypted key with $HMAC(k_e, nonce_e|k_h|P_h)$.

5. N_e verifies N_h and establishes a full link with N_h. Verifying $HMAC(k_e, nonce_e|k_h|P_h)$ convinces N_e that N_h holds k_e and enforces P_h. N_e overwrites P_e with P_h received at step 2. N_e then decrypts $ENC(k_e, k_h)$ and obtains k_h. It can now enable link layer encryption and establish full connectivity with N_h.

In case of $v_h < v_e$, the roles of N_e and N_h are just swapped. N_h sends v_h instead of P_h at step 2. N_e sends P_e at step 3 with $HMAC(k_h, nonce_h|P_e)$. The rest is similar.

Security Analysis. As discussed in EIAP, the attacker is unable to break the connection manager, the policy enforcer, or the trusted agent. As a result, she cannot obtain the link layer session key without enforcing the policy. Additionally, she cannot modify the policy or steal the link layer session keys on the machine either. So the new key distributed at step 4 is safe.

The number of old keys kept on each node, V, is a design parameter. If a node has missed too many policy updates, it cannot re-connect to the network through ERAP since none of the nodes enforcing the latest policy hold the old keys any more. In this case, it has to join as a new node through EIAP.

4.3 Limitations

Runtime intrusion. The security of our method largely depends on the security of the underlying trusted system, which only prevents untrusted code from being loaded from the disk. Therefore, our approach is unable to tackle runtime intrusion exploiting code vulnerabilities such as buffer overflows. This problem is mitigated because the attestation may reveal the code that has known vulnerabilities. Hence, the user can avoid trusting the vulnerable code. Furthermore, a successful intrusion is usually followed by invoking other local programs. Our method restricts the attacker's capability to run arbitrary local code because any code run by the protected service must be defined in the commitment.

Attacking the uncontrolled port. Neither EIAP nor ERAP prevents the attacker from sending link layer frames to the uncontrolled port of its one-hop away neighbor. The attacker can leverage this weakness to flood its neighbor nodes. We address this weakness from two perspectives. First, flooding the uncontrolled link layer port is by far less effective than flooding the network layer. This is because in the former case the attacker can only target a small number of nodes in her one-hop vicinity. In the latter case, she can target any node in the network and exploit widely distributed denial-of-service slaves to dramatically amplify the damage. Hence, our method addresses the main and most severe threat. Second, this weakness can be effectively addressed by host based countermeasures [11]. For instance, limiting the rate of accepting connection requests can foil resource depletion attacks.

Attacking the connection manager. The attacker may flood the connection manager of the protected node with overwhelming connection requests. To mitigate the problem, the protected node can limit the rate of handling the connection requests to bound the resources spent on EIAP and ERAP processing. In addition, EIAP and ERAP require the connection requester to do the attestation first at step 3. Since attestation is much more costly than verification, the difficulty of the attack is increased. The attacker has to either use a high-end computer or control a large number of low-end computers.

Kernel update. During the time the node maintains a link with the network, loading new kernel modules is limited by the kernel commitment. For instance, if the node obtains a new driver, which is not defined in the kernel commitment, it cannot load the driver until it reboots the system.

5 Case Study

To illustrate our mechanism, let us consider a simple ad hoc network created for a peer-to-peer file sharing application, namely Mute [5]. For instance, a group of students on-campus can use their 802.11-enabled smart phones to create such a network. The students do not know each other and want to protect themselves from being attacked by malicious peers. Since the network is formed for a specific application (i.e., Mute), the network access control policy must deny any other connection request from different applications. Additionally, even though Mute connections are accepted, the policy must protect the nodes against attacks that target Mute such as flooding.

Figure 5 shows an example of a policy for this application expressed in pseudo netfilter rules. We assumed that Mute runs on TCP port 5000. Furthermore, to allow multihop communication the network runs the AODV [21] routing protocol on UDP port 654. Each computer uses interface `wifi0` to join the ad hoc network.

```
IN
ℛ1   Mark=1 TCP from any to wifi0 !Local_IP:5000

OUT
ℛ2   Allow TCP SESSION from wifi0 to any:5000
ℛ3   Allow TCP SYN from wifi0 to any:5000 limit 3/s
ℛ4   Allow UDP from wifi0 to any:654 limit 10/s
ℛ5   Allow TCP from wifi0 to any:5000 Mark==1
ℛ6   Drop TCP or UDP from wifi0 to any

FORWARD
ℛ7   Drop from any to wifi0
```

Figure 5. The Policy for File Sharing Network

The rules $\mathcal{R}2-6$ allow only Mute and AODV traffic to be sent from each node. Hence, attacks to other services (e.g., default services such as `netbios`) are impossible. The attacker could try to exploit one of the allowed services: Mute and AODV. A direct attack is to flood Mute, but this is significantly limited by $\mathcal{R}3$ which allows a node to initiate at most 3 TCP connections per second. Once a session has been established, $\mathcal{R}2$ allows packets to be sent at any rate. Similar to $\mathcal{R}3$, $\mathcal{R}4$ protects AODV from being flooded. All these attacks are stopped at their originators without having any impact on the target node or the network. This is impossible through receiver side protection.

$\mathcal{R}1$ and $\mathcal{R}5$ enforce hop-by-hop packet forwarding for Mute traffic. Per $\mathcal{R}1$, the node marks an incoming (pre-routing) Mute packet of which it is not the destination. This means another node uses it as a router. Per $\mathcal{R}5$, the marked packets are allowed to be forwarded. In a more complicated scenario, some protected nodes may have multiple wireless network cards and be in multiple networks. The attacker may leverage this fact to launch attacks from an unprotected network to nodes in the protected network. $\mathcal{R}7$ stops this attack by forbidding packet forwarding across different networks.

Policies such as the one described in this section can be enforced by built-in kernel filters (e.g., netfilter). These filters can easily be extended by adding modules and thereby can support more complicated policies. For example, if the application is vulnerable to a buffer overflow attack and the attack signature is known, one can implement an extended module to check the specific signature in the packet and stop the attack at the originator. Consequently, our method is flexible and extensible.

6 Prototype Implementation and Evaluation

To show the feasibility of our mechanism, we implemented and evaluated a prototype that works over IEEE 802.11-based networks. We implemented the Satem prototype under the Linux 2.6.12 kernel [28]. The core of the prototype is the trusted agent, which is small but integrated in many places in the OS kernel, including system and kernel calls such as `do_execve`, `do_mmap`, `sys_init_module`, `sys_open`, `do_fork`, `file_nopage`, etc. We patched the official Linux kernel to add these modifications.

The EIAP and ERAP protocols are implemented in the connection manager as extensions of IEEE 802.1x. To implement the connection manager, we modify `xsupplicant` [6], an open source 802.1x client and `hostapd` [3], an open source 802.1x server. The two connection managers conduct the protocol negotiation via the `EAPOL` protocol [8]. We used the built-in netfilter as the enforcer.

To simplify the implementation, we used 802.11 WEP [2] to encrypt the link. Extensive research studies showed that WEP is insufficient to guarantee secure association [12]. Other types of stronger encryption such as WPA [2] can be used to replace WEP. For instance, integration with WPA is straightforward, but more configuration work in `xsupplicant` and `hostapd` is needed.

Our method incurs a certain latency for link layer connection establishment and data communication. To measure them, we used two nodes that create an IEEE 802.11b ad hoc network for the application described in Section 5. The source node N_S is an IBM R40 laptop with a 1.3Ghz Pentium M CPU, 512M RAM, an Intel IPW2100 wireless card, and an Atmel TPM. The destination node N_D is an IBM T43 laptop with a 1.7Ghz Pentium M CPU, 512M RAM, and an Atheros wireless card.

Scenario	Latency (in seconds)
802.11 WEP	1.2
EIAP	3.1
ERAP	1.9

Table 1. Connection Establishment Latency

Scenario	Download Speed (KB/second)
Open	235.23
WEP	230.57
Our Solution	229.42

Table 2. Data Communication Performance

The link establishment latency is shown in Table 1. Compared to the link layer connection establishment in the standard 802.11b with 104 bit WEP authentication, EIAP incurs a high overhead in the initial link establishment. This is mainly due to the cost of collecting attestation reports and the negotiations over `EAPOL`. As expected, the overhead of ERAP is significantly reduced because it does not perform the costly trust verification. We believe that these results are acceptable, especially because the high overhead imposed by EIAP is just a one-time cost. After that, the typical overhead is reduced through the use of ERAP for reconnections.

The overhead of joining is insignificant for the overall wireless communication performance because connection establishment happens only once in a while even in a volatile network. The cost that dominates the overall network performance is the latency of data communication. To quantify this cost, we measured the download speed of Mute in three networks: standard open 802.11b, standard 802.11b with WEP, and a trusted network that enforces the policy presented in Figure 5. Since we measured the cost when the link was fully established, this cost is relatively fixed per packet (i.e., policy enforcement). Therefore, using large files increases the accuracy of the cost estimation. In the test, we let node N_S download 256M files from N_D.

As Table 2 shows, our method incurs small performance degradation compared to both the open 802.11b (2.47%) and WEP-based 802.11b (0.5%). This result is due to the fact that our method does not incur any costs besides WEP encryption and packet filtering, which are very light-weighted. Another reason is the simplicity of the policy being enforced.

7 Discussion

Users in MANETs may want to update the network policy dynamically or to run multiple applications. Furthermore, mobility could lead to broken links between members of a protected network. This section discusses the behavior of our method when such situations happen.

Policy update In relative static ad hoc networks, there may be a need of updating the policy after a while; this issue can hardly appear in highly mobile networks which are short-lived by definition. The process of generating the new policy is the same as creating the first policy discussed in Section 3. When a node N_i updates its current policy with the new policy, it also replaces its old link layer key with the new key. If N_i is connected to N_j that has not updated its policy, the link is dropped. N_i can re-establish the link with N_j using ERAP protocol, causing N_j's policy to be updated. Therefore, updating the policy is the same as creating a new protected MANET. The only difference is that nodes join the new MANET through the much lightweight ERAP protocol.

Dropping the link will break any established application sessions. To mitigate this problem, we introduce a "non-disturbance" flag to the connection manager. The user can turn it on to defer policy update.

Execution of multiple applications. When the nodes of a protected MANET want to run a new application in addition to their existing applications, they need to update their current policy to cover the new application as well. By doing so, these nodes form a new protected MANET and disconnect themselves from other nodes in the existing protected MANET that refuse to update the existing policy.

The users may want to regulate each application. For example, nodes communicating through a file sharing application want to ensure that everyone provides a fair service (e.g., allowing other nodes to download files from them rather than just downloading from others). This requires defining a policy specific to each application and enforcing all policies simultaneously. Our method lacks this capability. Currently, we are investigating how to extend the idea of trusted MANET to multiple applications and develop a policy enforcing framework to support dynamic composition of trusted application networks.

Mobility. Mobility of nodes has negative impact on our method during link establishment phase. It may make the EIAP or ERAP protocol incomplete due to loss of physical connectivity. Hence, in order to join a trusted network, a node may have to try multiple nodes until it can complete the protocol with one of them. Once the trusted network is fully established, mobility does not affect our method. In this case, the overhead incurred from re-routing and retransmission is increased because of higher chance of broken links. However, since all nodes have the network key, they can establish a link between each other without going through our protocols. Hence, the only cost of our method in communication is enforcing the policy, which is determined by the total volume of traffic and the complexity of the policy.

8 Related Work

Our work leverages previous research on trusted computing and distributed policy enforcement.

Trusted Computing. Both hardware and software based methods have been proposed to ensure trusted software execution. The hardware approaches such as IBM 4758 [7], Dyad [30], and XOM [17] demand high-end hardware, which is unlikely to be ubiquitously deployed. Pure software methods such as [15], SWATT [25], and Pioneer [24] challenge the target system to attest its software stack within a time limit. These methods assume knowledge about the target system's clock speed and a noticeable delay caused by forging the same checksum. Neither of them holds in ad hoc networks computing.

As an emerging trend, Terra [13], Microsoft NGSCB [19], and IBM TCGLinux [23] balance between the hardware and software approaches. These approaches leverage a low-end trusted hardware like TPM [26] to boost trust on a set of software components, which further ensures trustworthiness of the execution of target programs. Satem differs from them in the scope and persistence of protection. First, our method only focuses on the code that the target programs depend on. Thus, it can catch every attempt to compromise the execution of the protected programs without false positives since irrelevant changes in the system will not be monitored. Second, by enforcing the commitments, our method guarantees the trustworthiness of the target programs not only at the time of attestation, but also for future executions.

Distributed Policy Enforcement. The distributed firewall concept originally proposed by Bellovin was implemented in [14]. In this approach, the firewall function is distributed on all protected hosts, each of which enforces the access control policy. This approach is a receiver side defense and does not prevent attacks from reaching the victims.

Our method is close to [22], which enables network access policies to be enforced on each VPN client. This method targets clients owned by legitimate users but improperly configured. As an attestation-only approach, it is insufficient to ensure trusted enforcement of the policy in face of malicious host owners. Our method, on the other hand, works even if the node is controlled by attackers. Furthermore, due to the ad hoc network nature, our method addresses unique issues such as policy synchronization, mobility, reconnections, and link layer integration, which do not exist in corporate VPN applications.

9 Conclusions

This paper presented a mechanism for creating protected ad hoc networks. The creation of such networks is triggered by users that want to run a common application. Our mechanism does not allow untrusted nodes to establish wireless links with nodes in the protected networks. Furthermore, it enforces a common network access control policy in the network; this policy is associated with the application running in the network. Attacks from member nodes are suppressed locally by the common network policy. To ensure trusted enforcement of the policy, we augmented every node with a trusted kernel agent based on the TPM. We evaluated the method through a prototype based on an IEEE 802.11 ad hoc network. The results demonstrate the feasibility of the proposed method as well as its low overhead.

References

[1] http://www.et2.tu-harburg.de/fleetnet/.

[2] 802.11 protocols. http://www.ieee802.org/11/.

[3] Hostapd. http://hostap.epitest.fi/.

[4] IEEE 802.1X Port-based Network Access Control. In *IEEE Standard 802.1X, 2001 Edition.*

[5] Mute. http://mute-net.sourceforge.net/.

[6] Open1x. http://open1x.sourceforge.net.

[7] Building a high-performance, programmable secure coprocessor. *Comput. Networks*, 31(9), 1999.

[8] Extensible Authentication Protocol Over Lan. In *IEEE Standard EAPOL*, 2000.

[9] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *Proceedings of Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, 2004.

[10] W. Bagga, S. Crosta, P. Michiardi, and R. Molva. Establishment of ad-hoc communities through policy-based cryptography. In *the Proceedings of Workshop on Cryptography for Ad hoc Networks (WCAN'06)*, 2006.

[11] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the 8th USENIX Security Syposium (Security'03)*, 2003.

[12] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. *Lecture Notes in Computer Science*, 2259, 2001.

[13] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[14] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a distributed firewall. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'00)*, 2000.

[15] R. Kennell and L. H. Jamieson. Establishing the genuinity of remote computer systems. In *Proceedings of USENIX Security Symposium (Security'03)*, 2003.

[16] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *the Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP'01)*, 2001.

[17] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, 2000.

[18] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of IEEE Symposium on Security and Privacy (S&P'00)*, 2000.

[19] Microsoft Corp. Next generation secure computing base. http://www.microsoft.com/resources/ngscb.

[20] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode. Trafficview: traffic data dissemination using car-to-car communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3), 2004.

[21] C. E. Perkins, E. Royer, and S. R. Das. Ad hoc on demand Distance Vector(AODV) routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, 1999.

[22] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of ACM Conference on Computer and Communications Security (CCS'04)*, 2004.

[23] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of USENIX Security Symposium (Security'04)*, 2004.

[24] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. V. Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP'05)*, 2005.

[25] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT:Software-based attestation for embedded devices. In *Proceedings of IEEE Symposium on Security and Privacy (S&P'04)*, 2004.

[26] Trusted Computing Group. TCG 1.1b Specifications. https://www.trustedcomputinggroup.org/home.

[27] Trusted Computing Group - Mobile Phone Working Group. Use Case Scenarios - v 2.7.

[28] G. Xu, C. Borcea, and L. Iftode. Satem: A Service-aware Attestation Method Toward Trusted Service Transaction. In *the Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, 2006.

[29] G. Xu and L. Iftode. Locality driven key management for mobile ad-hoc networks. In *the Proceedings of the 1th IEEE International Conference on Mobile Ad-hoc Networks and Sensor Systems (MASS'04)*, 2004.

[30] B. Yee. Using secure co-processors. Technical report, Carnegie Mellon University, 1994. PH.D Thesis.