

Secure Extension to the OLSR protocol

Andreas Hafslund, Andreas Tønnesen, Roar Bjørgum Rotvik, Jon Andersson and Øivind Kure

Abstract—In this paper we present a mechanism for securing the OLSR protocol. This mechanism is implemented as an extension to the OLSR source code provided from UniK (www.olsr.org), and can be downloaded freely from the Internet. The proposed and implemented mechanism is based on signing each OLSR control packet with a digital signature for authenticating this message. Also, the mechanism provides a timestamp exchange process. The timestamps are used to prevent replay attacks on the routing protocol. This security mechanism does not need synchronized time.

Index Terms—Mobile Ad Hoc Network, Security, OLSR, Digital Signatures

I. INTRODUCTION

A mobile ad hoc network (MANET) [1] is usually considered to be a network consisting of mobile nodes with wireless network interfaces. Each node can function both as an end-host, but also as an intermediate router for other nodes in the network. The mobility of the nodes makes the network topology dynamic.

Today wired computer systems can be made secure to a high degree, but when it comes to wireless networks weak security is often used if any security measurements are taken at all. This affects the services running on wireless networks including MANET routing protocols.

The OLSR [2] protocol is the current target for this investigation. We have made an implementation of a system for improving the security of the OLSR protocol with digital signatures in the routing messages. The implementation is an extension to the OLSR protocol, and secures only the routing messages itself, not the user traffic being routed in the MANET. Also the solution only provides integrity and not confidentiality, although the solution is extendable and could include mechanisms to provide confidentiality at a later stage.

This paper is divided into 5 sections. In section 2 we outline our proposed mechanisms for improving the security of OLSR, whereas Section 3 briefly considers some implementation issues. We discuss related work in Section 4, and conclude this paper in Section 5.

II. SECURING THE OLSR PROTOCOL

In this study we have chosen a security mechanism based upon signing each OLSR control packet with a digital signature for authenticating the messages. The digital signature is based on symmetric keys.

A. Overview

All OLSR control traffic is signed for every hop. This means that one does not have to consider variable fields in messages, such as hop count and TTL. It also means that only one signature is needed, although several OLSR messages are stacked in one OLSR packet. Using this hop-by-hop approach does not provide end-to-end signatures, which again means that the digest is not a true signature with respect to the

originator, but rather a signature from the forwarder, ensuring that it trusts the source of the message in the previous hop.

A node that does not have access to the shared secret key cannot produce a verifiable digest. All receivers running secure OLSR discard messages with non-verifiable digests. Signatures are transmitted in OLSR messages of their own. This is to ensure compatibility with nodes not running secure OLSR, and because a timestamp is transmitted in addition to the signature.

Four different messages are defined. One, which is the actual signature message, as displayed in Fig. 1, and three messages (Fig. 2 to Fig. 4) used in the timestamp exchange. All the messages are sent as the message body of an OLSR message.

To prevent *replay attacks*, timestamps are used in our secure extension to OLSR. To exchange these timestamps upon initial connection between two nodes, a two-way timestamp exchange mechanism is utilized. The solution *does not rely on synchronized time*.

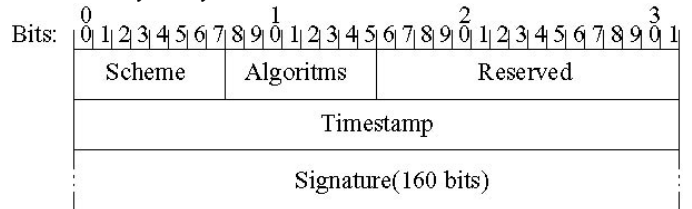


Figure 1: The basic signature message.

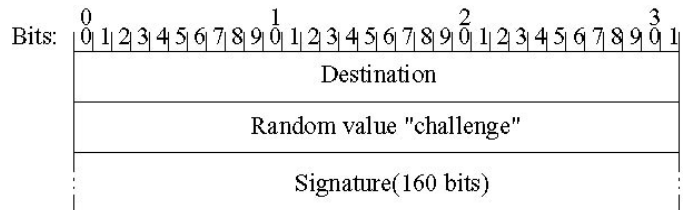


Figure 2: The timestamp exchange challenge message.

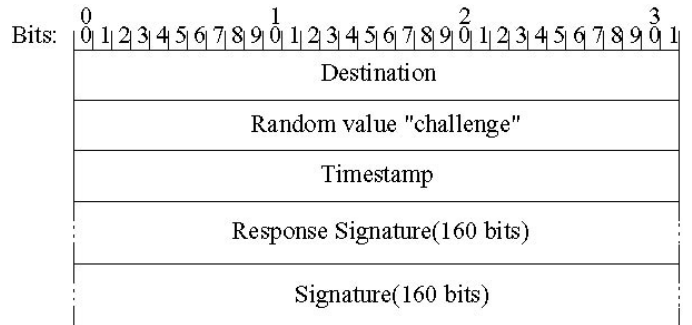


Figure 3: The timestamp exchange challenge-response message.

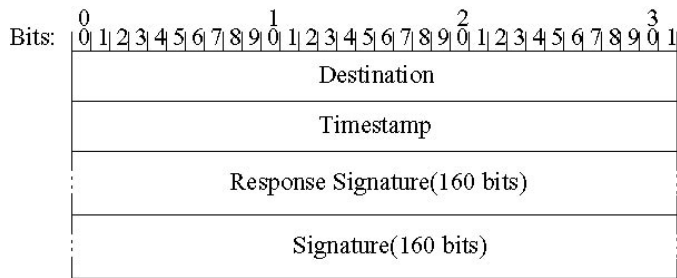


Figure 4: The timestamp exchange response-response message.

B. Signature

The signature message illustrated in Fig. 1 is attached to all outgoing OLSR packets. This message is to be the last message in the packet. The OLSR packet header is adjusted to include the size of the signature message in the size field.

The *scheme* and *algorithms* fields in the signature message header, informs the receiver of what signature scheme and what algorithms is being used. The *timestamp* field is for preventing replay attacks. The digest used as a signature is a hash created using the SHA-1 hashing algorithm, which produces an irreversible 160-bit digest. The hash is based on the following:

- The OLSR packet header (with adjusted size).
- All OLSR messages in the packet except the signature message.
- The OLSR message header, the sub header and the timestamp of the signature message.
- The shared secret key.

No considerations have to be taken regarding the variable fields of different headers (such as TTL or hop count) as this signing is done hop-by-hop.

The timestamp approach needs some further study, especially considered for time synchronization problems, MAC delay etc.

C. A flexible solution

Secure OLSR is designed to be as flexible as possible with regards to the encryption and hashing algorithms being used, and the entire signature scheme. In the implemented solution digests created by the SHA-1 hashing algorithm, of the message and a shared secret key is used to ensure integrity.

A different scheme could include one signature for each message allowing for end-to-end signing or asynchronous or synchronous encryption could be used to insure confidentiality as well. These schemes could again utilize different algorithms for hashing and encryption all being defined by the signature message header.

D. Timestamps and freshness

An attacker can record signed traffic and play it back at a later stage (replay attacks). This can be prevented to some degree by sequence numbers, which are already utilized in OLSR. However, for traffic that is only to be sent one hop, like HELLO messages, this is of little or no help. An attacker can simply record all messages transmitted by a node, and then move to another area of the network where the HELLO messages recorded was never heard. Here the attacker can

start the replay attack by transmitting the recorded messages. The OLSR sequence numbers are also weak because of their length. They are only 16-bit values and wrap-around* will occur rather frequent. The wrap-around mechanism used in OLSR makes the signature numbers even weaker with respect to freshness.

E. Timestamps exchange

The timestamp exchange process introduces three new message types. These messages are processed regardless of the signature message validation, since this process is likely to take place between neighbors that have no registered timestamp of each other. Because of this, all such messages are signed internally. This means that all the messages carry their own digest.

The exchange of timestamps between two neighbor hosts A and B can be described as:

$$\begin{aligned}
 &A \quad B : Ch_a D(M, K), \\
 &B \quad A : Ch_b Ts_b D(IP_b, CH_a, K) D(M, K), \\
 &A \quad B : Ts_a D(IP_a, CH_b, K) D(M, K).
 \end{aligned}$$

When A receives a signed message from a neighbor B for which A has no registered time value, A initiates the timestamp exchange process. A first sends a challenge message (Fig. 2) to B. This message is broadcasted since A might not have an actual route to B. The challenge message contains the IP address of B and a 32-bit nonce (number used once), Ch_a . This is a random number, which is used to append random data to real data to prevent a replay attack. A then signs this message with a digest of the entire message and the shared key $D(M, K)$.

B now has to respond to this message with a challenge-response message. B first generates the digest of its IP address (if B is multi-homed, the IP address fetched from the challenge message used), the received nonce and the shared key $D(IP_b, CH_a, K)$. B then generates a 32-bit nonce and transmits the IP address of A, the nonce, the timestamp of B, the digest $D(IP_b, CH_a, K)$ and a digest of the entire message and the shared key $D(M, K)$.

When A receives the challenge-response message from B, it first tries to validate the data. If the digests $D(IP_b, CH_a, K)$ and $D(M, K)$ can be verified, then the timestamp of B is used to create the difference of time between A and B. A then generates a response-response message (Figure 5) and broadcasts it to B. This message contains the IP address of the receiver (B), $As\ timestamp$, a digest of As address (as received from B), the nonce received from B, the shared key $D(M, K)$ and a digest of the entire message and the key $D(M, K)$.

When B receives the response-response message from A, it tries to verify the digests. If they can be verified then B uses the received timestamp to register its time difference to A. And the timestamp exchange is complete.

The solution does not require synchronized *time* but the clocks are assumed to be relatively synchronized, meaning that they are running on a relatively equal frequency. All timestamps are represented with a 32 bits value containing

* When a number stored in a variable is increased from the max allowed, the size of the data type, it will *wrap around* and start over at 0.

seconds since the epoch⁴. Timestamps are then at first recorded as $T = T_L - T_R$, where T_L is the local timestamp and T_R is the remote timestamp received through the timestamp exchange.

When receiving a signature message a certain slack S in the calculated timestamp difference is allowed. So that a signature message with a verified digest and a timestamp difference T_N , so that $T_O - S < T_N$ and $T_O + S > T_N$, where T_O is the stored timestamp difference of the sender, is considered a verified signature message.

To compensate for a possible skew between clocks, the timestamp difference is recalculated for every received and verified signature message. The difference is recalculated as $(T_O + T_N)/2$, where T_O is the recorded timestamp difference and T_N is the difference calculated based on the received timestamp.

F. Robustness

The timestamp exchange process could be exploited by an adversary to create an overload of processing and network usage, which could lead to the node not being able to participate in other timestamp exchanges or perhaps any communication at all. This would be a typical Denial of Service (DoS) attack. An attacker or a badly configured host could for instance transmit thousands of the timestamp exchange challenge messages within a very short period of time, all aimed at the same host. This would cause the receiving host to generate and transmit signed replies to all the challenges.

To avoid this a timer is set for the originators of all received challenges. Any new received challenges from the same host while the timer has not timed out, are discarded. Due to the signing of the challenge messages, an attacker cannot spoof the sender address of challenge messages. An attacker could however record all challenge messages directed to a host for a long period of time and launch them all within a short period of time. However, as timestamp entries are cached within nodes, this amount of messages would not be extensive.

III. IMPLEMENTATION ISSUES

Our secure OLSR proposal is implemented as an olsrd plugin. The UniK OLSR plugin is described in [3]. The implementation includes message signing and timestamp exchange, and is part of the olsrd source code available for download at www.olsr.org.

A. Overview

Implementing functionality that was to work on all incoming and outgoing “raw” OLSR traffic required an extension to the network output functioning in olsrd. An overview of the relations between the plugin and olsrd is illustrated in Fig. 5. The implementation is to be as transparent to the olsrd code as possible. Therefore all incoming traffic is passed to the plugin, which verifies the packet and removes the signature message and updates the

size field of the OLSR packet header. For outgoing traffic the opposite goes. All outgoing OLSR traffic is passed to the plugin, which adds the signature and updates the packet size.

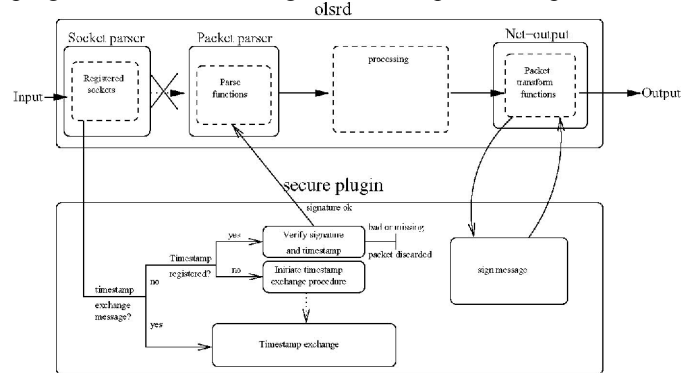


Figure 5: An illustration of the design of the secure plugin, as related to olsrd.

B. Shared Key

The key used is read from the file `/root/.olsrd/olsrd-key`. It is 128-bits of size. If no key can be read from the file, the plugin will terminate the olsrd process with a warning message. Other solutions might be implemented in future versions to handle local key management better or to be able to work in an integrated fashion with some authentication scheme.

C. Intercepting incoming traffic

The secure OLSR plugin must be able to intercept all incoming OLSR traffic and check the signature if present. This is done quite easily due to the modular structure of olsrd. The plugin deregisters all the OLSR sockets from the socket listener, and then register them again with the plugin's own input function. This is implemented in `src/socket_parser.c`. The OLSR sockets can all be retrieved from the global interface list `ifnet`. The plugin's own OLSR input function keeps the registered message parser functions and only differs from the original input function in olsrd on two points: (i) An incoming packet is checked for timestamp exchange messages, which are processed before the signature check. Keep in mind that these packets contain signatures of their own. (ii) An incoming packet is checked for an ending signature message. If no such message is found the packet is not considered sane and is discarded. If a signature message is received from a neighbor for whom no timestamp is registered, the timestamp exchange process is initialized. If the neighbor is registered the signature is checked. If the signature cannot be verified, the packet is discarded. If the signature is verified the timestamp is checked. If the timestamp validates the packet is passed on to the packet parser within olsrd.

D. Intercepting outgoing traffic

The plugin also needs to be able to intercept all outgoing traffic to add signature messages. To be able to do this a new set of function pointers was added to olsrd. They are called `packet transform` functions. A plugin can register its own packet transform functions with olsrd, and these functions are applied to every OLSR packet to be transmitted right before sending it. It is guaranteed that no changes will be made to a

⁴ The time and date corresponding to 0 in an operating systems clock and timestamp values. Under most Unix versions, the epoch is 00:00:00 GMT, January 1th, 1970.

OLSR packet after these functions are called. The function to add such a function pointer is implemented in *src/net.c: int add_ptf(int (*)(char *, int *));*

The plugin registers a function that calculates and adds a signature to the end of all outgoing OLSR packets. To be sure the packet will have room for the signature message (especially when stacking messages) the maximum message size in olsrd is set to *maxmsgsize - sizeof(signature msg)*.

E. Timestamp exchange

The Secure OLSR plugin maintains a repository of registered timestamps. Whenever a node receives a packet containing a valid signature message for which it has no timestamp entry registered the timestamp exchange process is initiated. All timestamp messages are sent as broadcast within regular OLSR packets. The messages use the OLSR message header. All incoming OLSR packets are checked for such messages before the signature check. This way these messages are not discarded even if they carry a timestamp from a non-registered host. The packets are however passed on for verification even though they only contain the timestamp exchange message. This is to prevent link hysteresis to consider these packets as lost if the node is verified. A scenario where a host, which the receiving host already has done a timestamp exchange with, can arise if the remote host has restarted olsrd, are timed out its timestamp entry for the receiving host

IV. RELATED WORK

The work [4] has a similar approach for securing the OLSR protocol as described in this paper. As our approach, [4] uses digital signatures for authenticating OLSR messages. Also, they propose a timestamp mechanism against replay attacks, and outline two public key infrastructure systems for MANETs. Even though our approach is similar to [4], there are some important differences.

In [4] they propose to include one signature for each OLSR message, not one for each OLSR packet. In addition to this, one timestamp is provided for each signature. The timestamps are for estimating the freshness of the messages. Thus avoiding replay attacks. The signature is encapsulated and transmitted as an ordinary OLSR message. In contrast to our approach, the signature message in [4] does not need to be in the same packet as the message it is a signature for. This means that the signature and the message can travel in separate packets and separate routes from the originator. Also, the proposed system in [4] is an end-to-end system, whereas our approach is a link-based system.

The timestamp exchange protocol proposed in [4] is a rather complex solution. It relies on a *timestamp exchange message* being periodically transmitted by each node. The timestamps are not considered fresh until the *handshake* is complete.

A more thorough discussion of the advantages/disadvantages between the security mechanism proposed in [4] and our proposed system is left for further study.

V. CONCLUSION AND FURTHER STUDY

We have designed and implemented a security extension for the OLSR protocol. The extension is tested and the correctness of the implementation is verified. However, this is not included in this paper.

Our solution adds extra overhead to all OLSR packets. The extra overhead is relative to the size of the OLSR packets, since the signature size is static. The larger the OLSR packets, the secure plugin will cause less effect on overhead. If using IPv6 addresses, this relative increase in overhead will be even smaller.

The implemented solution uses shared (symmetric) keys for signature creating and verification, and it is assumed that this key is accessible for all (trusted) hosts intended to be part of the MANET. Our secure extension to OLSR does not intend to cover key exchange/management or initial authentication. However, the secure extension is intended to be a part of a larger security scheme that does also cover these aspects. This is left for further study.

REFERENCES

- [1] J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations". IETF RFC 2501, January 1999.
- [2] T. Clausen, P. Jacquet, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot, "Optimized Link State Routing Protocol". IETF RFC 3626 October 2003.
- [3] A. Tønnesen, A. Hafslund, Ø. Kure "The UniK OLSR plugin". Currently under review for OLSR Interop and Workshop, 2004.
- [4] T.H. Clausen, C. Adjih, P. Jacquet, A. Laouiti, P. Muhlethaler, D. Raffo "Securing the OLSR Protocol". In Proceedings of IFIP Med-Hoc-Net, June 2003.