Analyzing the TLS protocol with Murphi
Project for CS513

Tim Wellhausen
Department of Computer Science
University of British Columbia

Version: December 4, 1998

1 Introduction

During this project I tried to prove the correctness of a networking protocol called TLS. My goal was to find out whether it could be possible for an intruder to get secret information sent between a client and a server. I used Murphi a program that explores the complete state space to find any conditions under which given invariants fail. To do this I had to model the protocol described and specified in an internet draft.

2 The TLS protocol

The TLS (Transport Layer Security) protocol is a further development of the SSL (Secure Socket Layer) protocol developed by Netscape Communications Corp. Thus, it is based mainly on SSL which has been widely used for about three years.

The purpose of this protocol is to create a secure connection between two computers over an insecure network. It uses the TCP protocol for a reliable connection and is based on top of it. It is quite easy to integrate into an application, because the protocol is capable of establishing and handling a secure connection completely on its own.

SSL is implemented in all important web browsers. The purpose of TLS is to replace SSL because SSL still has some unsolved security issues.

The most important part of TLS is the handshake between a client and a server to which the client wants to connect. During this handshake all security parameters and secrets are exchanged. Thus, this is the most vulnerable point of the protocol.

My work is based on the internet draft 5 by the Internet Engineering Task Force. This draft was replaced just some days ago by a new draft with minor changes.

Figure 1 shows how a connection is established. At first the client sends a ClientHello message to the server which includes the desired cipher suite and a random value. The

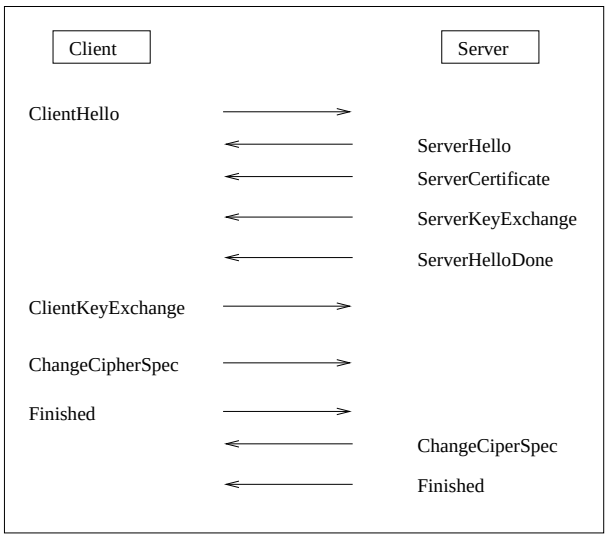

Figure 1: Message flow for a handshake

server has to respond with a ServerHello message which includes the chosen cipher suite, a session ID and another random value. The cipher suite determines which algorithm should be used for the key exchange and the encryption.

Dependent of the chosen cipher suite, the server sends a certificate and parameters for the key exchange. Afterwards a ServerHelloDone is sent to indicate that the server is finished.

Then the client has to send its parameters for the key exchange. Afterwards a ChangeCipherSpec message is sent, followed by a Finished message which is now encrypted with the key just exchanged. After this is done by both sides, the handshake is finished. Now both client and server are in possession of a master secret with which all following messages, the application data, are encrypted.

3 Murphi model of TLS

My model of this protocol is based on the work of Ulrich Stern's Murphi model of the Kerberos protocol. I used the basic structures and the same notation he used in his work.

The model implements the behavior both of the client and server while trying to connect. Furthermore there exists an intruder which has the capability of overhearing and manipulating all messages in the network.

The protocol allows the use of different methods for the key exchange. The standard methods include the use of RSA or Diffie-Hellman. I implemented only the key

exchange with RSA.

It is assumed that the cryptographic algorithms used by the protocol are safe. Furthermore it is assumed that it is not possible to attack a certificate.

4 Results

After implementing the exchange of all messages between client and server I had to realize that it seems not to be possible to attack this protocol with Murphi as long as certificates are used. If RSA is used, the public key of the server is part of the certificate. Suppose that the certificate cannot be changed by an intruder (this could also be a subject for proving), then the client gets the authenticated public key of the server which it uses to send a pre-master secret. Together with the random values exchanged in the beginning, the master secret is computed.

The draft describes in detail how a possible attack with replayed or generated messages will fail because of the exchange of specific messages. For example an encrypted MD5 checksum of all handshake messages is sent in the end.

I tried to implement an intruder that is able to store and send messages that he overhears. But because of the large number of messages, this blows up for one client and one server even with the hash table compression and a lot of memory.

Finally I decided to reproduce a possible attack that is described in the draft. This attack is only possible if client and server agree on a cipher suite that doesn't demand server authentication. That means there are no certificates involved.

Without authentication a man-in-the-middle attack is possible. The intruder has to intercept the ServerKeyExchange witch contains the public RSA key of the server and has to send a new ServerKeyExchange message with his own RSA key to the client. When the client sends the pre-master-secret which is now encrypted with the intruder's key, the intruder has to intercept again and send the pre-master-secret with the server's public key to the server.

Together with the random values which the intruder can overhear, it can easily compute the master secret without anybody knowing that there is an intruder. For this the intruder has to keep track of all messages, and to manipulate the MD5 checksum at the end of the handshake.

My implementation of this attack consists of an intruder that overhears all important information and generates new KeyExchange messages. I assume that the intruder would be able to send only messages if they would be accepted by the destination. This is possible if the intruder stores information about all messages sent. I skiped this in order to avoid the next blow-up. Furthermore, I didn't model the exchange of the correct checksum in the end.

The following is an extract of Murphi's output for this attack:

```
Startstate Startstate 0 fired.
```

```
Rule Client sends ClientHello to server.
Rule intruder overhears/intercepts.
Rule Server receives ClientHello from a client and sends ServerHello.
Rule intruder overhears/intercepts.
Rule Client gets ServerHello message.
Rule Server sends ServerKeyExchange.
Rule intruder overhears/intercepts.
Rule Intruder sends generated ServerKeyExchange message to client.
Rule Client gets ServerKeyExchange message from server.
Rule Server sends ServerHelloDone.
Rule Client gets ServerHelloDone message from server.
Rule Client sends the ClientKeyExchange message.
Rule intruder overhears/intercepts.
Rule Intruder sends generated ClientKeyExchange message to server.
Rule Server gets ClientKeyExchange message.
Rule Client sends the ChangeCipherSpec message.
Rule Server gets ChangeCipherSpec message.
Rule Client sends the Finished message.
Rule Server gets the Finished message and sends ChangeCipherSpec message.
Rule Client gets ChangeCipherSpec message from server.
Rule Server sends Finished.
Rule Client gets Finished message from server.

End of the error trace.

======================================================================

Result:
        Invariant "Intruder must not have the master secret
        of a running connection." failed.

State Space Explored:
        1161 states, 2759 rules fired in 0.65s.
```

I also tried this attack with two clients, two servers and one intruder. It took 246837 states until the same result was computed.

5 Conclusion

My personal summary is that at least for me it was not possible to find any bugs in this protocol. Furthermore I think that it would be very difficult for anybody to use Murphi to model the whole protocol. After reading the draft several times very carefully I understand how the protocol works. I guess that there are still some bugs

in it, but they'll probably appear only in situations that are too specific for Murphi to model.

One of the few changes in the draft just released addresses such a bug that was found recently. The following is the short description:

```
An attack discovered by Daniel Bleichenbacher can be used to attack
a TLS server which is using PKCS#1 encoded RSA. The attack takes
advantage of the fact that by failing in different ways, a TLS server
can be coerced into revealing whether a particular message, when
decrypted, is properly PKCS#1 formatted or not.
```

For me it seems like a long way to go until such bugs can be discovered with the help of an automatic prover.

Nevertheless I think that modeling the TLS protocol was a very good way to understand how this protocol actually works! And I believe that it is reasonable to use a program like Murphi at least for the design of a new protocol.

References

[SSL96] Netscape Communications Corp.: *The SSL 3.0 Protocol*, Nov 18, 1996

[TLS98] The Internet Engineering Task Force: *The TLS Protocol Version 1.0*, Nov 12, 1998, URL: http://www.ietf.org/internet-drafts/draft-ietf-tls-protocol-06.txt