Verification of the TLS handshake protocol using ProVerif

Freek van Walderveen

9th June 2008

1 Introduction

In this report I describe an implementation and verification of the TLS handshake protocol in ProVerif. The handshake protocol is, with regard to security protocols, the most interesting of the three protocols forming the TLS standard as it establishes the session keys used for further communication and allows for server and client authentication. My implementation is primarily based on Paulson’s paper Inductive Analysis of the Internet Protocol TLS with some details taken from RFC 4346. The model used is a simplification of the original protocol where some messages and message parts are left out that are not important for the proofs of the security goals. Like Paulon’s, the present implementation allows both client certificate verification and a simple kind of session resumption.

The following security goals are proved in the presence of internal threats and active attackers.

- Mutual authentication after the handshake is finished (in case the client certificate and certificate verify messages are sent, otherwise only the server is authenticated);
- the premaster and master secrets shared by two honest agents cannot leak, even if the session keys of an earlier incarnation of the current session are lost.

2 Informal protocol narration and implementation

The following informal protocol narration should give a better overview of the protocol model as implemented. Agent $A$ is a client with public key $kA$ and $B$ a server with public key $kB$.

$A \to B : A, na, sid, pa$

$B \to A : nb, sid, pb$

$B \to A : \text{cert}(B, kB)$

$A \to B : \text{cert}(A, kA)$

$A \to B : \{|pms|\}_pB$

$A \to B : \{\#(nb, B, pms)\}_sA$

$A \to B : \{\text{finished}\}_{\text{client}K_{(na, nb, m)}}$

$B \to A : \{\text{finished}\}_{\text{server}K_{(na, nb, m)}}$

where $\text{finished} = \#(sid, m, na, pa, A, nb, pb, B)$ and $m = \text{prf}(pms, na, nb)$

The certificate $\text{cert}(a, p)$ of the public key $p$ of an agent $a$ is implemented as signing $(a, p)$ using a certification authority’s private signing key. There is exactly one certification authority and anyone can get hold of its public unsigning key.
A resumption of the above session then proceeds as follows, assuming both parties agree on the reuse of the SessionID and secrets. Note that the participants do however generate new session keys from the fresh nonces.

\[
\begin{align*}
A \rightarrow B : & \quad A, na', sid, pa' \\
B \rightarrow A : & \quad nb', sid, pb' \\
A \rightarrow B : & \quad \{\text{finished}^{'}, \text{clientK}(na',nb',m)\} \\
B \rightarrow A : & \quad \{\text{finished}^{'}, \text{serverK}(na',nb',m)\}
\end{align*}
\]

where \( \text{finished}^{'} = \#(sid, m, na', pa', A, nb', pb', B) \)

In the ProVerif pi-calculus implementation, the client and server can resume a session indefinitely and always accept resumption. Although this is not very realistic, it captures all scenarios:

- If both the client and the server want to resume, the full path including session resumption can be followed by both the client and server agent;

- if, without loss of generality, the client does not want to resume, the session resumption will not take place as then the server runs a new instance (due to the replication) and generates a different SessionID not matching the corresponding input statement in the client’s resumption code. Now, the client can also choose to follow a non-resumption path because it can also spawn multiple sessions.

To summarize, the following parts are replicated in my model.

- The initializer, client and server processes, such that an attacker can communicate to an arbitrary number of agents to gain information for some attack.

- The code implementing one session (including resumptions) in both the client and the server.

- The code implementing one resumption in both the client and the server, such that sessions can be resumed endlessly.

There is however only one non-replicated compromised agent process, so an attacker can use only one certified key pair.

2.1 Non-deterministic choice

The handshake protocol as described in RFC 4346 allows the server to ask the client to authenticate herself. My initial implementation (tls.pi) enforces this, making the implementation simpler, hence easier to understand and quicker to verify. I have however also implemented a simplified certificate request/server hello done message (tls_nd.pi) that allows the server to make the choice whether the client should send her certificate and proof of ownership. This choice should be made non-deterministically such that both options are verified. One, but most likely not the only, way to model non-determinism in ProVerif/pi-calculus is to use ambiguous destructors. In combination with an encoding for “yes” and “no” respectively as \((())\) and \(()\), I used this construct to model the server’s non-deterministic choice to ask the client for its certificate. The server sends an extra message to the client to notify her of this decision and expects the client to follow.
3 Verification

My model queries for the following properties.

1. Each time a client receives a valid **finished** message from an honest server, she knows that this server indeed just sent a corresponding one.

2. If a client sent a valid **certificate verify** message in the current session, then each time a server receives a valid **finished** message from this client, he knows that this client indeed just sent a corresponding one.

3. An attacker cannot get hold of the premaster or master secret of a session between two honest agents.

These properties guarantee the following security goals.

1. After a completed handshake, the client knows that the server is who he claims to be and that he indeed wants to communicate with her in this session.

2. If the client is honest and authenticated, after a completed handshake the server knows that the client indeed wants to communicate with him in this session.

3. The session keys remain safe as they are based on the master secret and there is no way of obtaining them other than through the \texttt{clientK()} and \texttt{serverK()} constructors. Furthermore, if the session keys are lost after a handshake, a resumption of the same session will still be secure. The loss of session keys is modelled by letting the client publish these keys after each completed handshake.

4 Discussion

Apart from the usual abstractions such as the assumption of perfect cryptography and simplifications of the messages, my model does not directly verify the following properties.

- Secrecy of the session keys.
  Only the underlying master secret is guaranteed to be safe. However, by inspection of the protocol one can observe that this implies that the session keys will also remain safe, except for the deliberate publishing by the client to model the loss of session keys.

- Authentication under the presence of multiple certification authorities.
  This should however also not be a big problem, as one could see this one certification authority as an imaginary root CA that trusts all of the other certification authorities.

- Secrecy under the variant of TLS that does not use server authentication.
  This variant would not require much work to implement, but it is not often used in practice and simplifies the implementation.

- Secrecy under the variant of TLS that uses Diffie-Hellman to compute a premaster secret.

ProVerif did not find any attacks against my model, but I do not want to draw any further conclusions.